UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Big Data aplicado al Marketing

Trabajo de fin de grado

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

Realizado por:

Aleix Rauet Garcia

Supervisado por: Oscar Esparza

Barcelona, Junio de 2019

Abstract

Being one step ahead of the Market has always been a goal for any company.

This thesis aims to make predictions about the profile of customers and future customers of a brand in the automotive industry.

These predictions are intended to make personalized offers to the future customer before the purchase is made or to make suggestions to the customer to encourage new purchases.

Several models will be proposed for each case presented, and finally the scores obtained in each case will be compared to determine which one fits better.

Resum

Anar un pas per davant del Mercat sempre ha estat objectiu per a qualsevol empresa.

En aquesta tesi es pretén realitzar prediccions sobre el perfil dels clients i futurs clients d'una marca del sector de l'automoció.

Amb aquestes prediccions es faran ofertes personalitzades al futur client abans que realitzi la compra o bé, es faran suggeriments al propi client per incentivar noves compres.

Es plantejaran diversos models per a cada cas plantejat, i finalment es compararan les puntuacions obtingudes en cada cas per determinar quin s'ajusta millor.

Resumen

Ir un paso por delante del Mercado siempre ha sido un objetivo para cualquier empresa.

En esta tesis se pretende realizar predicciones sobre el perfil de los clientes y futuros clientes de una marca del sector de la automoción.

Con estas predicciones se pretende hacer ofertas personalizadas al futuro cliente antes de que realiza la compra o bien, hacer sugerencias al propio cliente para incentivar nuevas compras.

Se plantearan varios modelos para cada caso planteado, y finalmente se compararán las puntuaciones obtenidas en cada caso para determinar cuál se ajusta mejor.

Agradecimientos

Antes de nada, me gustaría dedicar esta tesis a mi familia.

Además, agradecer la confianza de Jaume López por permitirme hacer esta tesis en Deloitte, a mis compañeros Arnau Arnaiz y Alex Buxó por ayudarme e inspirarme a plantear casos que resultaran útiles para el proyecto.

También, agradecer a mi amigo Pol Casals por haberme introducido en esta empresa y animarme a plantear la tesis dentro del entorno laboral.

Agradecer también a mi compañero de laboratorio David Fité, el cual entendió mi situación e invirtió más horas en el laboratorio de las que le tocaban.

En último lugar, pero menos importante, a Oscar Esparza el cual pese a apenas conocerme y al plantearle una tesis que en primera instancia no resultaba clara, decidió aceptar ser mi tutor.

Historial de revisiones

Revisión	Fecha	Propósito
0	08/06/2019	Creación del documento
1	14/06/2019	Revisiójn de la primera parte de la memoria
2	24/06/2019	Revisión de la segunda parte de la memoria

Lista de distribución del documento

Nombre	e-mail
Aleix Rauet	aleixrauet@gmail.com
Oscar Esparza	oscar.esparza@entel.upc.edu

Escrito por: Aleix Rauet Garcia		Revisado por: Oscar Esparza	
Fecha	25/06/2019	Fecha	25/06/2019
Posición	Autor	Posición	Supervisor

Indice

Lista de Ilustraciones

Lista de Tablas:

1. Introducción

Tener información relevante y robusta que permita anticiparse a los movimientos de los futuros o propios clientes resulta una gran ventaja competitiva, que tendrá un significativo impacto en la captación o fidelización de estos.

Actualmente se dispone de una gran cantidad de información, pero hacer que el impacto de toda esta información sea positivo a nivel de mejora sigue siendo un reto.

Utilizar grandes bases de datos para obtener información relevante es el objetivo de este trabajo de fin de grado, aplicando el Big Data al marketing de un fabricante de automóviles.

El término Big Data resulta muy amplio, de tal manera que seré más específico a la hora de determinar qué se realizará.

Entendemos por Big Data a los conjuntos de datos o combinaciones de datos cuyo tamaño, complejidad y velocidad de crecimiento dificultan su procesamiento. Utilizaremos esta gran cantidad de datos para realizar predicciones, modalidad conocida como Machine Learning.

Si estas predicciones son suficientemente exactas y se adaptan al mercado actual serán utilizadas para para crear una experiencia de cliente personalizada para cada perfil.

1.1. Propósito

Antes de determinar los objetivos, realizaremos una breve introducción a la manera de trabajar de esta marca para entender mejor el propósito de la tesis.

La obtención de clientes mayoritaria se realiza mediante marketing digital, siendo sus principales fuentes:

1. Anuncios en redes sociales
2. Anuncios en Internet
3. E-mail Marketing (Para clientes)

Si la persona está interesada en la compra de un vehículo y accede al anuncio, es redirigida a una página dónde se completa un formulario en el que se aportan datos personales tales como nombre, apellidos, email, número de teléfono, provincia…

Este último paso solo lo realizan las personas realmente interesadas en la compra. El concesionario más cercano contactará con la persona en cuestión para realizarle una oferta.

El propósito de las predicciones, será:

1. Hacer énfasis en aquellas personas que a priori no están tan interesadas en el vehículo
2. Determinar aquellos clientes que no pasarán por un taller oficial de la marca según su perfil
3. Crear sugerencias personalizadas según el perfil de la persona

1.2. Especificaciones y requerimientos

El objetivo de la tesis es crear una experiencia de cliente más personalizada, así como determinar en qué clientes se debe hacer foco. Esta información también tendrá repercusión en el dimensionamiento del stock de la marca. Para realizar lo comentado:

- La precisión de los modelos predictivos deberá ser mayor del 90%. Si las puntuaciones son inferiores serán orientativas
- El modelo se deberá entrenar diariamente para que las predicciones se adapten al máximo al perfil de la gente interesada

1.3. Metodología

A continuación, describiremos el proceso que se deberá seguir cuando se desarrolle un modelo:

1. Extracción de los datos: Se realizará una “query” de SQL para cada caso.
2. Limpieza de datos: Se prepararán los datos de manera que puedan ser procesados por los modelos sin errores y sin información redundante que pueda afectar al entreno del modelo
3. Entreno del modelo: Se procesarán los datos mediante una función de coste la cual adaptará el modelo a los datos
4. Evaluación de la puntuación del modelo: Se estudiarán los diferentes hiperparametros y se analizarán las muestras más allá de la puntuación para determinar la validez de la predicción

1.4. Plan de trabajo

Se pueden diferenciar 3 grandes bloques en el desarrollo de esta tesis:

- Iniciación y Fase de investigación: Durante este tiempo se estudiaron todas las posibilidades de realización, así como escoger el lenguaje de programación más adecuado y aprenderlo.
- Fase de implementación: Se determinaron los casos a estudiar y se plantearon los modelos que a priori se podrían adaptar mejor.
- Fase de testeo: Se testeo la precisión de cada modelo con diferentes variantes e hiperparametros.

NAME	BEGIN DATE	END DATE
Initiation		
Learn about the project	07/03/2019	21/03/2019
Revise the work done until my initiation	07/03/2019	21/03/2019
First contact with the Data model and sql	07/03/2019	21/03/2019
Investigation phase		
Python learning	21/03/2019	20/04/2019
Machine learning ivestigation	21/03/2019	20/04/2019
Implementation phase		
Develop the code to predict the data	20/04/2019	20/05/2019
Understand the results given by the predictions	20/04/2019	20/05/2019
Testing phase		
Determine the accuracy of the predictions	20/05/2019	25/05/2019
Find better fitting models	25/05/2019	03/06/2019
Hiperparameter tuning	03/06/2019	10/06/2019
Impact of the predictions on clients	10/06/2019	20/06/2019

Tabla 1: Timing del plan de trabajo

1.5. Incidencias

Durante la realización de esta tesis ha habido un gran número de incidencias, a continuación, listaré las más relevantes:

- Base de datos utilizada: La base de datos utilizada ha dificultado mucho el avance de la tesis, ya que hay un gran número de campos mal informados, nulos o duplicados. Esto ha hecho que la mayor parte del trabajo se focalizara en la limpieza de estos. La puntuación de los modelos era en un principio baja, hasta que vi como el atributo 'Modelo' estaba mal informado tal que si el modelo era 'Huracan' en el atributo podía aparecer como 'HURACAN', 'huracán', 'huracan' de tal manera que se interpretaban 4 modelos en lugar de uno. Pasó en más atributos, pero eran menos relevantes.

- Cantidad de registros: Lo que a priori era una base de datos prometedora (en cuanto a número de registros se refiere) acabó siendo una base de datos justa. Esto pasó por los campos que eran nulos y registros duplicados. Por ejemplo, en una predicción, antes de limpiar la base partíamos con 300.000 registros. Eliminados todos aquellos registros duplicados y con campos nulos y mal informados el número final se quedaba en 17.000, número insuficiente de registros para realizar un entreno del modelo correcto.

- Atributos no numéricos: Los modelos no 'entienden' campos que no sean numéricos de tal manera que se deben tratar. En primera instancia, se realizó un 'one-hot encoding' pero en los arboles de decisión no funcionó correctamente. En una primera aproximación se podría pensar en asignar un número a cada atributo no numérico, pero a la práctica distorsiona la información del modelo. Finalmente, se utilizó un "encoding" de tipo "target" que resultó funcionar correctamente.

- Anteriormente a la realización de ésta tesis no sabía ni Python ni SQL. Ha sido un proceso de aprendizaje de estos dos lenguajes de programación, además del Machine Learning en sí. Esto dificultó un poco el desarrollo de la tesis, cometiendo errores en la sintaxis que impedían avanzar al ritmo que a veces se esperaba.

2. Estado del arte sobre Machine Learning

2.1. Introducción al Machine Learning

El Machine Learning es la disciplina que proporciona a los ordenadores la capacidad de aprender por si mismos sin estar explícitamente programados para ello.

Un ejemplo de ello sería el filtro de Spam de cualquier aplicación de correo electrónico. El programa aprende a clasificar los mensajes en función de los "flags" de los usuarios.

Los ejemplos que el programa utiliza para aprender se denominan training data y para determinar la exactitud de éste se utiliza por ejemplo, el ratio de emails bien clasificados.

Procedamos ahora con un ejemplo más visual:

	f_1	f_2	...	f_K	y
x_1	$x_{1,1}$	$x_{1,2}$	...	$x_{1,K}$	y_1
x_2	$x_{2,1}$	$x_{2,2}$	...	$x_{2,K}$	y_2
...	...	...	...	...	...
x_r	$x_{r,1}$	$x_{r,2}$	...	$x_{r,K}$	y_r
x_{r+1}	$x_{r+1,1}$	$x_{r+1,2}$	...	$x_{r+1,K}$	y_{r+1}
x_{r+2}	$x_{r+2,1}$	$x_{r+2,2}$	...	$x_{r+2,K}$	y_{r+2}
...	...	...	...	...	...
x_N	$x_{N,1}$	$x_{N,2}$	...	$x_{N,K}$	y_N

Ilustración 1: Estructura de un set de datos para realizar Machine learning [9]

Como podemos observar la ilustración 1 sigue la estructura de una tabla, también conocido como "dataset", será el punto de partida de cada modelo predictivo. Está compuesta por K atributos y N registros. En la variable Y se almacena el resultado de la contribución de todos los atributos determinada por la ecuación $Y_1 = f((X_{1,1}), (X_{1,2}), \dots, (X_{1,k}))$.

El "dataset" está estructurado en dos partes:

- **Training set**: Compuesto por la parte de color azul, por convención suele ser el 80% del "dataset" y serán los datos que entrenen el modelo.
- **Test Set:** Compuesto por la parte tricolor del "dataset", serán los datos que utilicemos para realizar la medida de precisión de nuestro modelo.

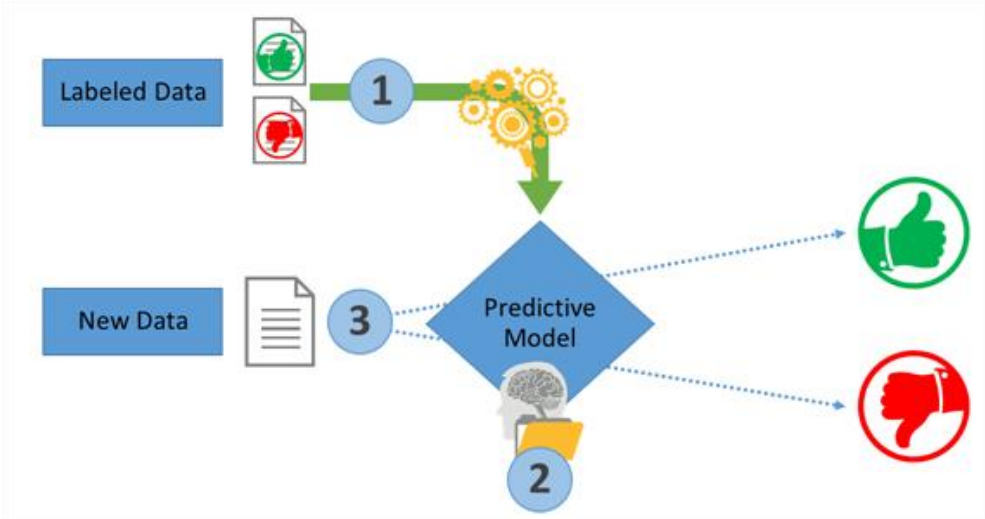

Ilustración 2: Estructura de training y validación [10]

En la ilustración 2 apreciamos el proceso de training y validación. Comentar que "labeled data" es nuestro training set, el cual tiene una Y asociada, mientras que "new data" es nuestro test set, del cual al modelo sólo le proporcionaremos la información sin la variable Y (a predecir por el modelo). Para determinar la precisión del modelo compararemos Y con $\bar{Y}$, la cual vendrá determinada por :

$$\bar{\boldsymbol{Y}} = \begin{pmatrix} f((X_{r+1,1}), & \cdots & f(X_{r+1,k})) \\ \vdots & \ddots & \vdots \\ f\left((X_{N,1})\right) & \cdots & f\left((X_{N,k})\right) \end{pmatrix}$$

2.2. Tipos de Machine Learning

A continuación, se describirán las 3 mayores agrupaciones de Machine Learning:

2.2.1. Clasificación mediante la cantidad y el tipo de supervisión que recibirán durante el entrenamiento:

- Supervised Learning: En este tipo de sistema el training set incluye el resultado (Función Y), llamado *Label.* Un ejemplo sería el comentado anteriormente del filtro de Spam, que está entrenado con multitud de e-mails flagueados por los usuarios y aprenderá a clasificarlos. Otro ejemplo podría ser el precio de un automóvil según su antigüedad, kilometraje, marca, modelo… Una vez el modelo estuviera entrenado sería capaz de predecir su precio en función de las variables de entrada.

A continuación, los modelos más relevantes que siguen este tipo de sistema:

- k-Nearest Neighbors
- **Linear Regression**
- Logistic Regression
- **Support Vector Machines (SVMs)**
- **Decision Trees and Random Forests**
- **Neural networks**

- Unsupervised Learning: Como bien se podía anticipar en este tipo de sistema el training set no incluye una variable resultante Y. El sistema trata de aprender sin ninguna variable en cuestión. Se suele utilizar en tareas de clasificación, y un ejemplo sería clasificar a los clientes de la marca de automóviles sin conocer el modelo de coche comprado.

  A continuación, los modelos más relevantes que siguen este tipo de sistema:

  Clustering

  - k-Means
  - Hierarchical Cluster Analysis (HCA)
  - Expectation Maximization

  Visualization and dimensionality reduction

  - **Principal Component Analysis (PCA)**
  - **Kernel PCA**
  - Locally-Linear Embedding (LLE)
  - t-distributed Stochastic Neighbor Embedding (t-SNE)

  Association rule learning

  - Apriori
  - Eclat

- Reinforcement Learning: Este caso es bastante distinto a los anteriores, el Sistema de aprendizaje llamado “agente” es capaz de observar el entorno, realizar acciones y recibir penalizaciones por las decisiones tomadas. El sistema debe aprender en función de estas penalizaciones y determinar la mejor estrategia para minimizarlas. Los algoritmos que utiliza un robot para aprender a andar son un ejemplo.

2.2.2. Clasificación en función de si el sistema puede aprender incrementalmente

- Batch Learning: Todos aquellos sistemas los cuales ante la aparición de nuevos datos se deben re-entrenar por completo. Esto a la larga puede suponer un problema de tiempo y poder de computación. También conocido como “off-line learning”.

- Online Learning: En este tipo de sistemas se entrena al modelo incrementalmente mediante pequeñas instancias o grupos llamados “mini-batches”. Suelen ser utilizados en aplicaciones como la predicción de los precios de las acciones, ya que hay que entrenar el modelo muy frecuentemente.

2.2.3. Clasificación en función de la generalización del modelo

Entendemos por generalización a cómo responde el modelo a inputs que no aparecen en el training set, una medida de calidad que hay que tener muy en cuenta a la hora de escoger el modelo, ya que poco tiene que con la precisión que obtengamos en el test set.

- Instance-Based learning: El sistema aprende de los ejemplos del training set como cualquier otro, pero si entra un nuevo registro que no aparece en el “training set” generaliza el caso usando una medida de similaridad. En el caso de predecir el precio de un vehículo utilizando su antigüedad, kilometraje, marca o modelo y apareciera un modelo en el “test set” no contemplado en el “training set”, el algoritmo buscaría la máxima similitud en marca, kilometraje y antigüedad.
- Model based learning: Una manera alternativa a la clasificación es crear un modelo de predicción en función de los atributos. Vamos a clarificarlo con una imagen

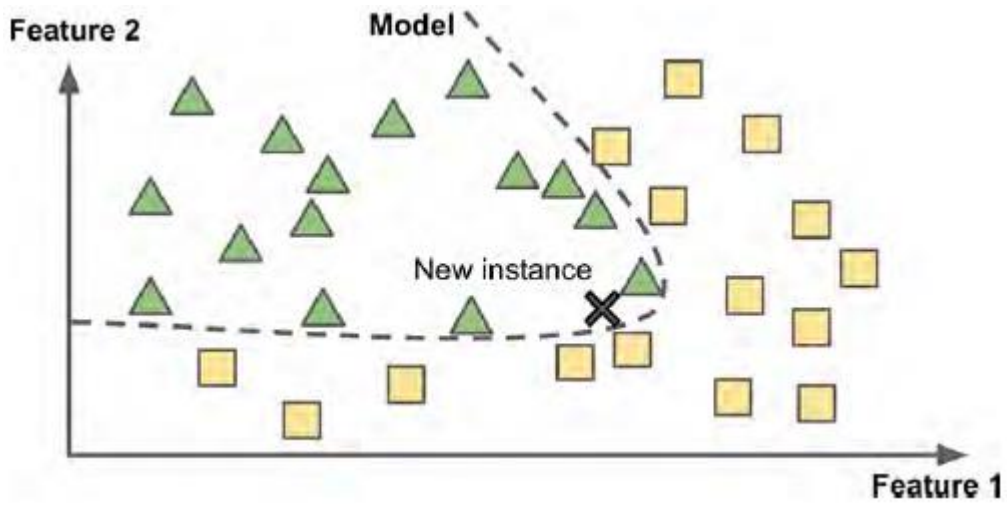

Ilustración 3: Ejemplo de un Model-Based Learning [4]

Como podemos observar hay una línea discontinua que delimita la relación entre los dos atributos, de tal manera que, aunque en el training set no aparezca, si está dentro de la superficie será interpretado correctamente.

2.3. Principales retos del Machine Learning

Hasta ahora hemos hecho una introducción al concepto Machine Learning y a sus principales tipos. Seguidamente describiremos las principales problemáticas a la hora de implantar soluciones basadas en Machine Learning

2.3.1. Insuficiente cantidad de datos para el Training Set

Para realizar un correcto entrenamiento y que el modelo se ajuste como es debido a los datos es necesaria una cantidad relativamente alta de datos. En aplicaciones sencillas con unos miles de registros será suficiente, mientras que para aplicaciones complejas pueden ser necesarios millones de registros.

Todo dependerá de la cantidad de atributos que se precisen para la predicción y de la correlación entre estos, como peor caso tendríamos un “dataset” con gran cantidad de atributos y poco correlados entre ellos.

En [1] Michele Banko y Eric Brill descubrieron la importancia del tamaño del “training set”. En este estudio se demostró que, si se dispone de una cantidad de datos suficientemente grande, la diferencia entre los modelos que se escoja es ínfima.

2.3.2. Datos no representativos, de poca calidad e irrelevantes

Para que el algoritmo generalice bien es crucial que el “training set” sea todo lo representativo de los datos como sea posible. De esta manera el modelo responderá correctamente a todas las entradas. Considerando el ejemplo anterior de la predicción del precio de un automóvil, si se utilizara un training set no representativo el modelo no realizaría predicciones acertadas en vehículos que resultaran ser muy caros o muy baratos.

Por lo que se refiere a los Datos de poca calidad, si el “dataset” está lleno de ellos el modelo tendrá mucha más dificultad para reconocer patrones. No suele ser un gran problema cuando no abundan en el “dataset”, estos datos son eliminados durante el pre-procesamiento.

El problema reside en cuando abundan, y reducen el “dataset” de tal manera que pierde precisión por falta de muestras.

En cuanto a la irrelevancia de los datos, nuestro sistema solo será capaz de aprender si los datos son suficientemente relevantes. En este ámbito es donde entra en juego la elección de los atributos. Una buena manera de escoger los atributos será computar las matrices de correlación entre ellos de tal manera que descartamos aquellos que no aporten información al sistema.

2.3.3. Sobre-entrenamiento y Sub-entrenamiento del Modelo

Hasta el momento, hemos visto como la abundancia de datos en el training set tiene un impacto muy positivo en el entrenamiento del modelo. Es totalmente cierto, pero como en todo, una enorme cantidad de datos hará que nuestro modelo funcione peor según [2]. Esto sucederá ya que el modelo se adaptará mucho al training set, de tal manera que puntuará peor en el test-set. Nos damos cuenta que hay que buscar un compromiso entre generalidad y especialidad en nuestros modelos para que funcionen correctamente

En el caso contrario y más lógico si cabe, el sub-entrenamiento del modelo hará que no se adapte lo suficiente a los datos y obtendremos una puntuación pésima.

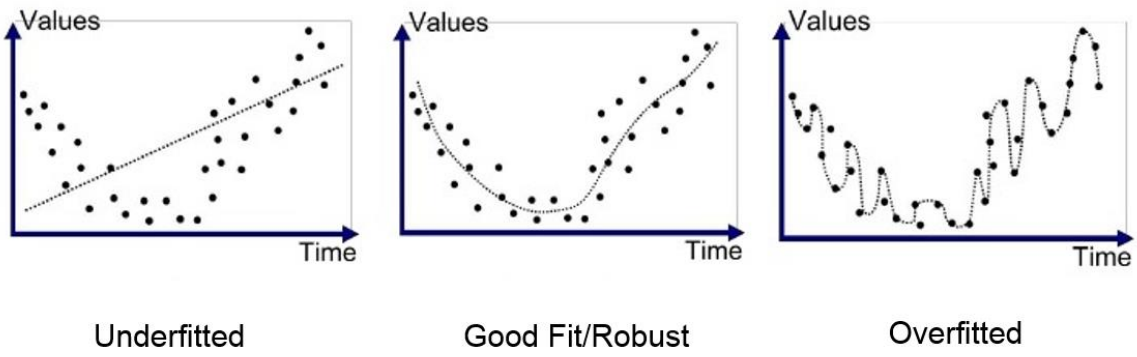

Ilustración 4: Ejemplo del "Fitting" en un Dataset [11]

En la figura superior apreciamos los tipos de "fitting" ante un modelo datos. Para aclarar más los conceptos supongamos que nos compramos un vehículo de una determinada marca y al poco tiempo se nos rompe por un defecto de fabricación. Si fuéramos personas "overfitters" diríamos que cualquier coche de esa marca se va a romper. En caso de ser "good fitters" entenderíamos que es un defecto de fabricación y no cuestionaríamos la integridad de la marca. Finalmente, si fuéramos "underfitters" no nos percataríamos de la avería (Resulta imposible).

3. Metodología / Desarrollo del proyecto:

A continuación, se detallará la metodología y el desarrollo seguido durante la tesis.

Ha Habido 3 casos donde se han aplicado varios modelos de Machine Learning y se ha comparado su puntuación final en función de cómo se adaptan al modelo.

- Conversión de persona interesada en un vehículo a cliente
- Listado los clientes con sus respectivas visitas al taller
- Listado de clientes con modelo de vehículo

En este apartado se discutirán los modelos utilizados, desde un punto de vista matemático y desde un punto de vista más enfocado al tipo de datos que disponemos.

Antes de empezar, detallar que todo el trabajo se ha realizado con PYTHON y su api SCIKIT-Learn específica para realizar Machine Learning la cual es ampliamente utilizada.

Se han utilizado datos desde 2018 por dos razones:

1. Si se hubieran utilizado los datos históricos no se hubiera podido entrenar los modelos por falta de poder de procesamiento (No se ha utilzado ninguna plataforma de Cloud Computing) todo el entrenamiento se ha realizado de manera local
2. Falta de generalidad: Hay gran cantidad de modelos los cuales no se fabrican, mientras que ha habido una gran incorporación de modelos estos últimos dos años. Trabajar con datos históricos no hubiera tenido ningún sentido y las puntuaciones hubieran sido muy bajas

Como paso común en los 3 casos tendremos la limpieza y tratamiento de datos, el cual será detallado a continuación.

3.1.1. Limpieza y tratamiento de los datos

Una vez se ha hecho la extracción mediante una “query” de SQL de la base de datos se debe pre-procesar para que el modelo pueda interpretar la información.

El resultado es un archivo .Csv el cual leeremos desde Python con la librería pandas y su función “read_csv”.

Una vez hemos importado los datos los tendremos en un “dataframe” de la librería pandas, procedemos a:

- Eliminar las filas en que haya algún atributo nulo
- Eliminar duplicados

Posteriormente se escogerán que atributos deberán contribuir al modelo, así pues se ejecutará una matriz de correlación para ver qué atributos contribuyen más

El siguiente paso será tratar los atributos no numéricos para que los algoritmos sean capaz de 'entederlos'.

Para dicha tarea se utiliza el método llamado "one-hot encoding" el cual binariza los diferentes campos del atributo.

No hay una integración total entre las librerías Pandas y SCIKIT-LEARN, de manera que a veces puede resultar que los datos se lean incorrectamente.

Siguiendo con el funcionamiento del "one-hot encoding", el algoritmo de transformación recorrerá el atributo a binarizar y realizará un conteo de las diferentes apariciones. Este número de diferentes apariciones determinará el número de bits del "encoding", el cual tendrá un solo uno por aparición. Veamos un ejemplo:

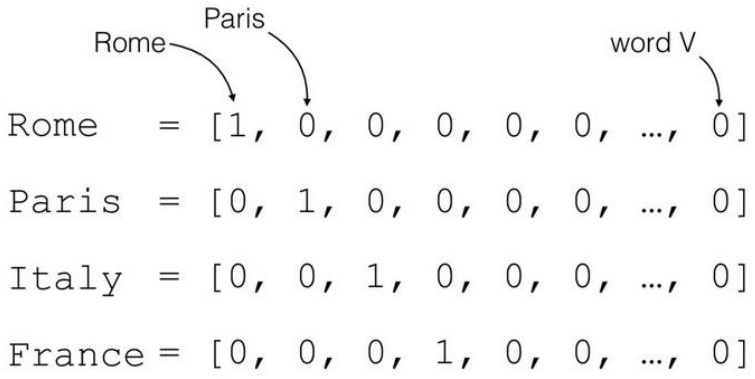

Ilustración 5 Ejemplo de OneHot Encoding [12]

Como podemos apreciar en la Figura 5, aparecen V palabras, de manera que tenemos V bits y un solo uno por palabra, determinado por su posición.

Como punto positivo, este método resulta ser muy simple, pero como punto negativo resulta ser muy ineficiente a nivel de memoria y la similitud entre palabras desaparece, de manera que el training set tendrá que ser muy completo y representativo si queremos tener una buena precisión. Más adelante se verán otros métodos de "encoding", pero este suele ser un buen punto de partida.

Seguidamente se hará un “reshape” del “dataset” para introducir estos atributos binarizados.

Esto se realizará haciendo que cada palabra del atributo pase a ser un atributo, y en el campo aparezca un ‘0’ o un ‘1’.

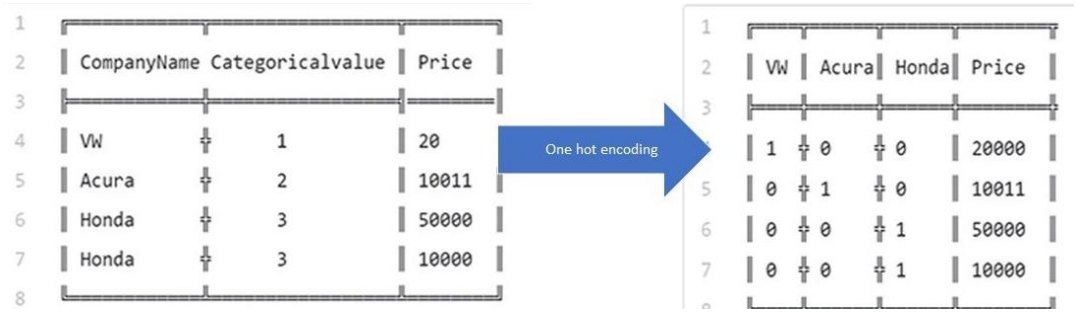

Ilustración 6: Inserción de las variables binarizadas en el ‘Dataset’

En el ejemplo anterior observamos como el atributo ‘CompanyName’ ha sido substituido por tantas columnas como diferentes palabras aparecían en este.

Finalmente, y en cuanto a pre-procesamiento se refiere, eliminaríamos todos aquellos duplicados que nos interesen.

3.1.2. Estructura del código

A continuación, se describirá brevemente la estructura del código utilizado en los 3 casos. Esta estructura determina los pasos que se han seguido desde la obtención de los datos hasta la puntuación final obtenida por el modelo.

Añadir que los modelos de SCIKIT-Learn siguen una estructura determinada, tal que los datos están organizados en “X”, la matriz de entrada o “feature matrix”, y en “y”, el cual es el vector de resultados. Damos por hecho que los resultados no están en la matriz de entrada

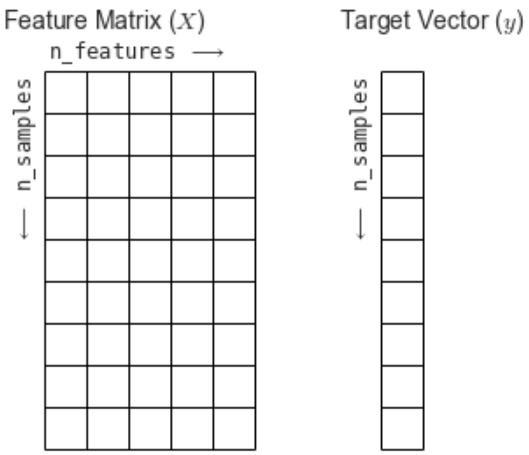

Ilustración 7: Estructura de los datos para cualquier modelo de Scikit-Learn

Pseudocódigo

ReadCsv

Data Cleaning

Optimizaciones aplicadas al modelo

Split del dataset en X,y

- *Split de X en X_train y X_test*
- *Split de y en y_train y y_test*

Instancia del modelo

Entreno del modelo con X_train y y_train

- *Predicción del Modelo entrenado con X_test (Obtención de $\hat{y}$)*
- *Puntuación del Modelo comparando y_test e $\hat{y}$*

3.1.3. Support Vector Machines

Una “Support Vector Machine” o SVM resulta ser una de los modelos más versátiles y capaces del Machine Learning. Para realizar su explicación utilizaremos imágenes ya que el concepto resultará más fácil de entender:

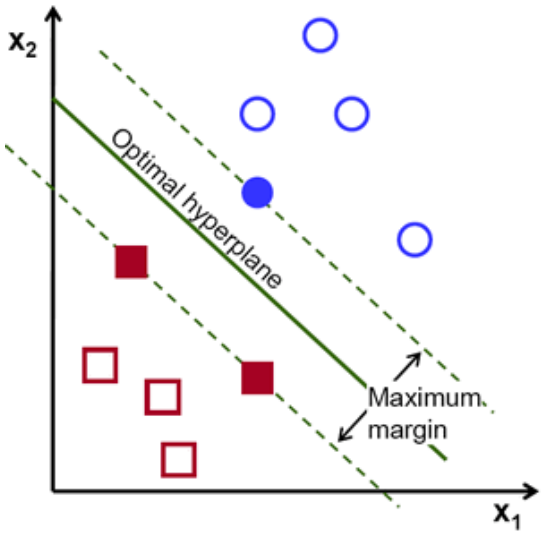

Ilustración 8: Clasificación mediante SVM [13]

En la ilustración 8 podemos observar como dados dos atributos $\{X1, X2\}$ se realiza una clasificación siguiendo la estructura de un vector, según un hiperplano o límite de decisión. El modelo de clasificación será más o menos restrictivo en función del margen que se imponga entre las dos muestras diferentes más cercanas.

Además de realizar clasificación también podemos realizar regresión con este modelo (Aplicable a los casos planteados). En vez de intentar establecer el mayor espacio entre las muestras diferentes como en la clasificación, en la regresión se trata de agrupar la mayor cantidad de muestras iguales limitando las violaciones de hiperplano. La anchura de este hiperplano viene determinada por el parámetro ϵ.

Este modelo realiza las predicciones según la función:

$$\hat{y} = \begin{cases} 0\ si\ \boldsymbol{w^t} * \boldsymbol{x} + b < 0 \\ 1\ si\ \ \boldsymbol{w^t} * \boldsymbol{x} + b > 0 \end{cases}$$

**Los elementos en negrita se interpretan como vectores*

Donde:

- $\boldsymbol{w}$ hace referencia a la influencia o peso de cada instancia
- b es una variable que introduce sesgo

Haciendo referencia a la figura 7, entendemos como el 0 de la función $\hat{y}$ pertenece a un cuadrado mientras que el 1 de la función $\hat{y}$ pertenece a un circulo. Con esto se pretende ejemplificar el resultado de la función.

Hemos determinado como realizar la función, pero ¿Cómo se determinan $\boldsymbol{w}$ y $\boldsymbol{b}$?

El propósito del entrenamiento de un modelo precisamente es determinar este tipo de constantes.

A continuación, se muestra la función a optimizar para determinar estos parámetros:

$$\frac{1}{2} * \boldsymbol{w^t} * \boldsymbol{w} + C * \sum_{i=1}^{m} \zeta^i$$

Sujetos a

$$t^i\left(\boldsymbol{w^t} * \boldsymbol{x^i} + b\right) \geq 1 \qquad para\ i = 1,2,\dots,m$$

Donde:

- t^i será -1 para instancias negativas y 1 para instancias positivas
- ζ^i es una variable la cual determina cuanto puede violar el margen de decisión l i-ésima instancia
- El parámetro *C* ayudará a determinar el "trade-off" entre minimizar la variable ζ^i y minimizar el producto vectorial entre $\boldsymbol{w^t} * \boldsymbol{w}$ que incrementará el margen de decisión

Dada la versatilidad de las SVM's, podemos realizar funciones polinómicas con ellas. En el ejemplo de la ilustración 8 se ha mostrado una SVM lineal.

Se define en términos de Machine Learning la función kernel, la cual es capaz de computar el producto entre dos transformaciones a vectores sin tener que necesariamente saber la transformación que se ha aplicado.

Definamos la función kernel como $K(\boldsymbol{a}, \boldsymbol{b}) = \phi(\boldsymbol{a}) * \phi(\boldsymbol{b})$.

Donde $(\boldsymbol{a}, \boldsymbol{b})$ son vectores de longitud n y ϕ una transformación aplicada a dichos vectores.

$$\begin{aligned}
\text{Linear:} \quad & K(\mathbf{a},\mathbf{b}) = \mathbf{a}^T \cdot \mathbf{b} \\
\text{Polynomial:} \quad & K(\mathbf{a},\mathbf{b}) = \left(\gamma \mathbf{a}^T \cdot \mathbf{b} + r\right)^d \\
\text{Gaussian RBF:} \quad & K(\mathbf{a},\mathbf{b}) = \exp\left(-\gamma \| \mathbf{a} - \mathbf{b} \|^2\right) \\
\text{Sigmoid:} \quad & K(\mathbf{a},\mathbf{b}) = \tanh\left(\gamma \mathbf{a}^T \cdot \mathbf{b} + r\right)
\end{aligned}$$

Figura 8: Transformaciones aplicadas a los casos de esta tesis

Una vez definidos estos kernels, podemos adaptar nuestra SVM a prácticamente cualquier forma.

3.1.4. Arboles de decisión

Los Arboles de decisión posiblemente sean el modelo más utilizado en el Machine Learning por ser uno de los algoritmos más potentes para realizar todo tipo de tareas.

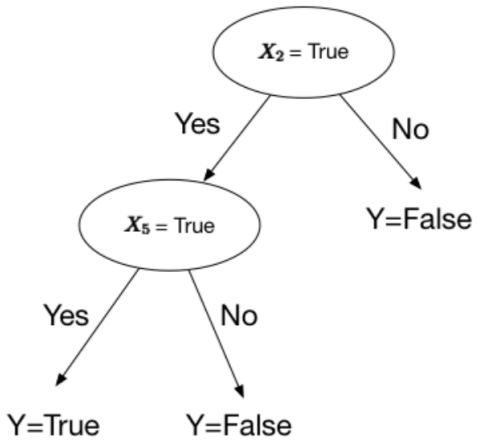

Ilustración 9: Arbol de decision

En la figura superior podemos observar la estructura que siguen los arboles de decisión. Se componen de un nodo principal o "root" del cual parten todas las decisiones. Posteriormente, se va ramificando en decisiones binarias hasta llegar a una decisión final.

Cuando se realice el entrenamiento del modelo, éste determinará la ubicación de los nodos. Cuanto más cerca del "root" se esté más importancia tendrá esa "leaf" o decisión intermedia en el resultado final.

Como el modelo nunca se comportará de manera perfecta, se puede cuantificar el error cometido en cada nodo con una medida de impureza llamada "Gini". En el ejemplo anterior es difícil de ver, pero si suponemos un árbol de muchos niveles y atributos de entrada, no todas las muestras de entrenamiento pertenecerán a la misma clase, de tal manera que cumplirán las condiciones de los nodos por los que hayan pasado pero los valores de sus atributos no serán 100% iguales.

Para calcular la Impureza Gini de un árbol de decisión se utiliza

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

Donde $p_{i,k}$ es el ratio de muestras clase K entre las muestras de entrenamiento del nodo i-ésimo.

Para realizar el entrenamiento, la API de SCIKIT-Learn sigue el algoritmo CART:

El algoritmo en primera instancia el training set en dos "subsets" utilizando un solo atributo k y un "treshold" t_k.

Para decidir que k y t_k utilizar, busca el par que reduce el MSE. La función de coste que el algoritmo trata de minimizar es:

$$j(k, t_k) = \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right} \text{ donde } \begin{cases} MSE_{node} = \sum_{i\ \epsilon\ node} (\hat{y}_{node} - y^i)^2 \\ \hat{y}_{node} = \frac{1}{m_{node}} \sum_{i \epsilon node} y^i \end{cases}$$

- $m_{left/right}$ es el número de muestras a la derecha/izquierda del subset

- $MSE_{left/right}$ mide el error a la derecha/izquierda del subset

La complejidad del árbol vendrá determinada en función de la profundidad. Es decir, si no limitamos la profundidad del árbol el algoritmo tendrá en cuenta todos los atributos y registros, dando lugar a un árbol de grandes dimensiones y sin introducir incertidumbre según [4]. En cambio, si esta profundidad se limita sólo se tendrán en cuenta aquellos atributos y registros más relevantes, si habrá cierta incertidumbre asociada y el árbol será de tamaño mucho más reducido. Limitar la profundidad también nos ayudará a reducir el 'overfitting'.

3.1.5. Ensemble Learning y Random Forests

Entendemos por "ensemble learning" todos aquellos modelos sujetos a una multitud de estimadores con diferentes parámetros donde la mejor precisión será la escogida. Estos modelos se basan en el concepto "Sabiduría de las masas" el cual determina que las opiniones de cientos de personas correctamente agregadas pueden superar el conocimiento de un experto en una materia determinada.

Por ejemplo, se podría entrenar varios árboles de decisión, cada uno con una parte diferente del training set.

Se obtendrían las predicciones de cada árbol individual, y finalmente se predeciría la clase que reciba más votos.

Este tipo de modelos se encontrarán siempre al final de los proyectos, una vez se tengan unos modelos lo suficientemente sólidos para combinarlos y obtener *El Modelo* de modelos.

Voting Classifiers

Supongamos que tenemos varios modelos ya entrenados para un determinado caso (Regresión lineal, SVM, SVM con un kernel Gausiano, Arbol de decisión…) los cuales tienen una precisión determinada no muy dispar entre ellos.

Una manera muy simple de crear un mejor modelo es agregar las predicciones de cada modelo y predecir la clase que recibe más votos

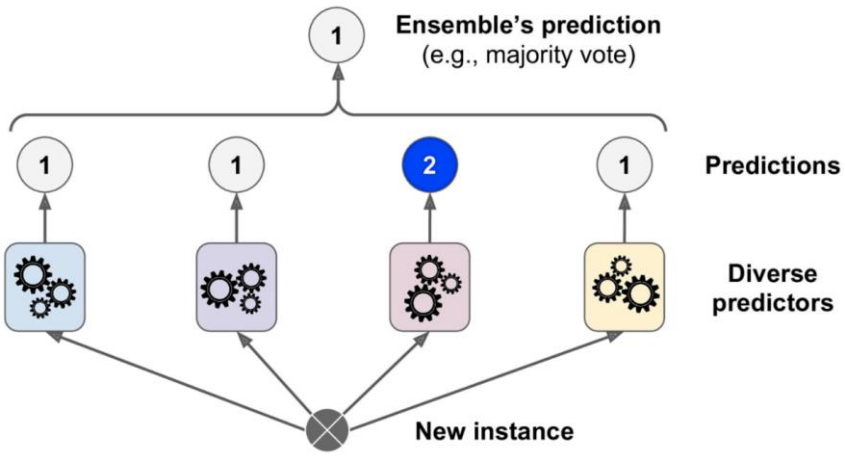

Ilustración 10: Ejemplo de un Voting Classifier [5]

En este caso muy probablemente obtengamos una mayor puntuación que el mejor modelo. Esto sucede gracias la ley de los grandes números, la cual establece que si tenemos, por ejemplo, 1000 predictores que individualmente son capaces de obtener una precisión del 51%, si predecimos con las clases que han recibido más votos se puede esperar hasta un 75% de precisión [5] .

De todas maneras, esto solo sería cierto en caso de que todos los modelos fueran independientes, resultando en errores incorrelados lo cual no resulta cierto en nuestros casos ya que los modelos estarían entrenados con el mismo training set.

Para realizar dicho Voting Classifier SCIKIT-Learn tiene la función VotingClassifier.

Pasting and Bagging

En este caso se entrenará a los modelos con diferentes partes del training set, donde se podrán repetir muestras de entrenamiento entre los diferentes modelos, pero solo se podrán repetir estas muestras en el mismo algoritmo si se utiliza el bagging. Este proceso se muestra en la ilustración 11.

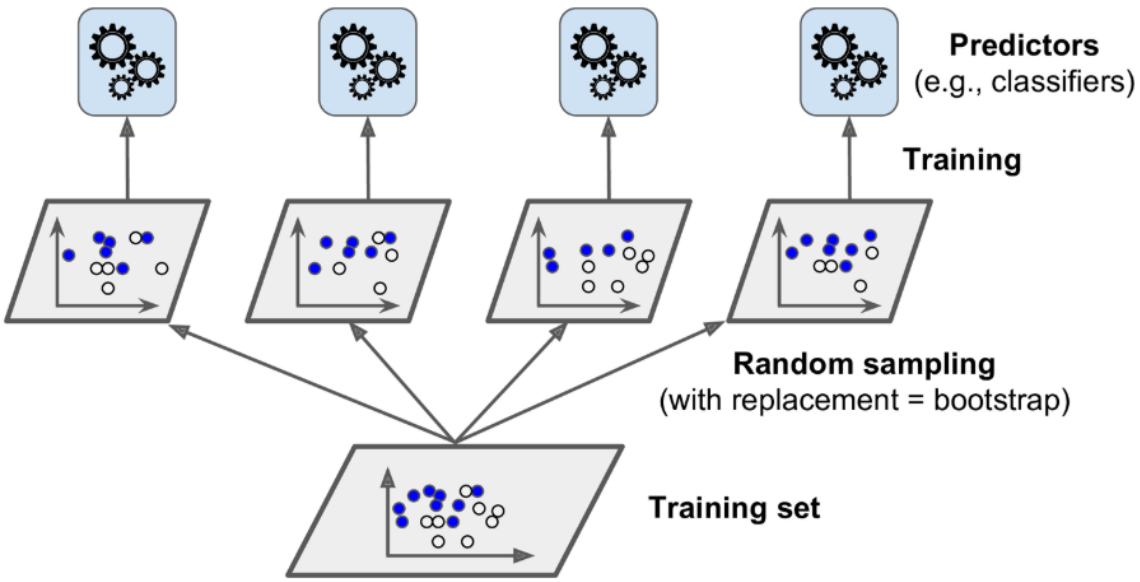

Ilustración 11: Aplicación del Pasting and Bagging en un training set con varios modelos [5]

Una vez los modelos están entrenados y mediante “ensemble voting”, se realizan las predicciones para una nueva entrada comparando las predicciones de todos los modelos.

Se puede implementar mediante la función de SCIKIT-Learn BaggingClassifier.

Random Forest

Como bien se podría deducir, un “ramdom forest” resulta ser la agrupación de árboles de decisión en los que se realiza “ensenble learning”. Generalmente se entrenan mediante el método de “Bagging”.

Los “Random Forest” suelen tener los mismos hiperparametros que los arboles de decisión además de los parámetros del “Bagging Classifier”.

Este algoritmo introduce un extra de aleatoriedad cuando va “creciendo” los árboles. En vez de buscar el atributo que más contribuye cuando se divide un nodo, busca el mejor atributo dentro de un seguido de atributos arbitrarios. Esto resulta en una mayor diversidad, lo cual se traduce en un mayor sesgo, pero en una mayor varianza.

Este modelo se puede implementar mediante la clase Random Forest de SCIKIT-Learn

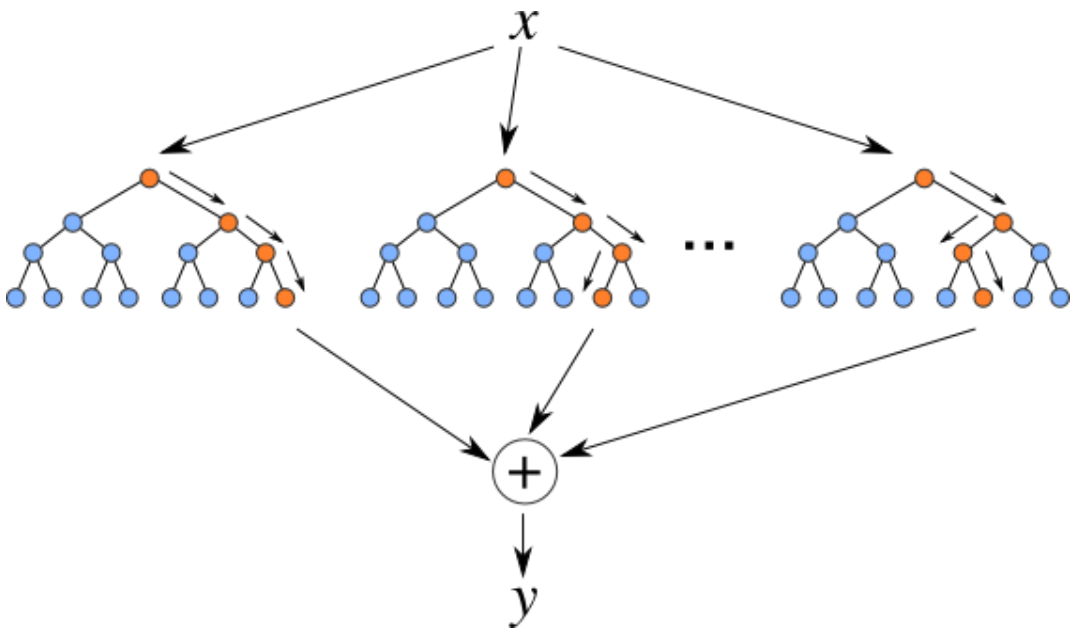

Ilustración 12: Random Forest [4]

4. Resultados

A continuación, se plantearán los casos y se mostrarán los resultados obtenidos. Remarcar que se ha realizado la elección de modelos en función de su precisión. Según [3], si no se asume nada sobre los datos no hay razón por la cual a priori preferir un modelo u otro. En este caso simplemente se han utilizado aquellos modelos que más precisión podrían aportar.

4.1. Conversión de persona interesada en el vehículo a cliente

El objetivo de este caso es poder determinar las personas que una vez han estado interesadas en el vehículo, han acabado comprándolo. De esta manera se puede hacer "push" desde la marca a aquellos que tienen un perfil menos comprador o bien, asegurar a aquellas personas que tienen un perfil más comprador.

En un contexto normal, ejecutaríamos una matriz de correlación y escogeríamos aquellos atributos más relevantes, pero en este caso y por imposición del fabricante se han escogido los siguientes atributos:

- **Código de persona:** Código numérico único e irrepetible para cada persona
- **Modelo:** Atributo no numérico que describe el modelo en el que se tiene interés
- **Origen del interés en el vehículo:** Atributo no numérico que determina el origen de la captación del cliente interesado. Hay varios orígenes como podrían ser Banners en redes sociales, publicidad on-line, eventos que realiza la marca…
- **Flag de compra:** Atributo numérico que determina si ha habido compra

- **Provincia:** Atributo no numérico que indica la provincia dónde reside el cliente
- **Comunicaciones Comerciales:** Atributo que determina si el cliente puede recibir publicidad
- **Gama:** Atributo no numérico que indica la gama del vehículo (comercial, turismo..)
- **Año del modelo:** Atributo que indica el año del vehículo
- **Numero de opcionales:** Atributo que indica el número de opcionales del vehículo
- **Combustible:** Atributo no numérico que indica el tipo de combustible
- **Edad:** Atributo que indica la edad del cliente
- **Modelo del vehículo:** Atributo no numérico que indica el modelo del vehículo

Se parte de un “dataset” con 313.000 registros, el cual después de ser pre procesado eliminando registros nulos y duplicados se queda en 273.000. A priori parece un número más que suficiente para entrenar cualquier modelo dada la poca cantidad de atributos.

La puntuación del modelo se obtiene con la función AccuracyScore, la cual comprueba muestra a muestra el vector predicción y del de test y da un porcentaje de acierto.

El primer modelo planteado para este caso será una *SVM*:

Para realizar correctamente la SVM y dado que nuestros datos no son de valor binario, se deberá normalizar los valores de los atributos de tal manera que estén representados en la misma escala para conseguir una mayor precisión

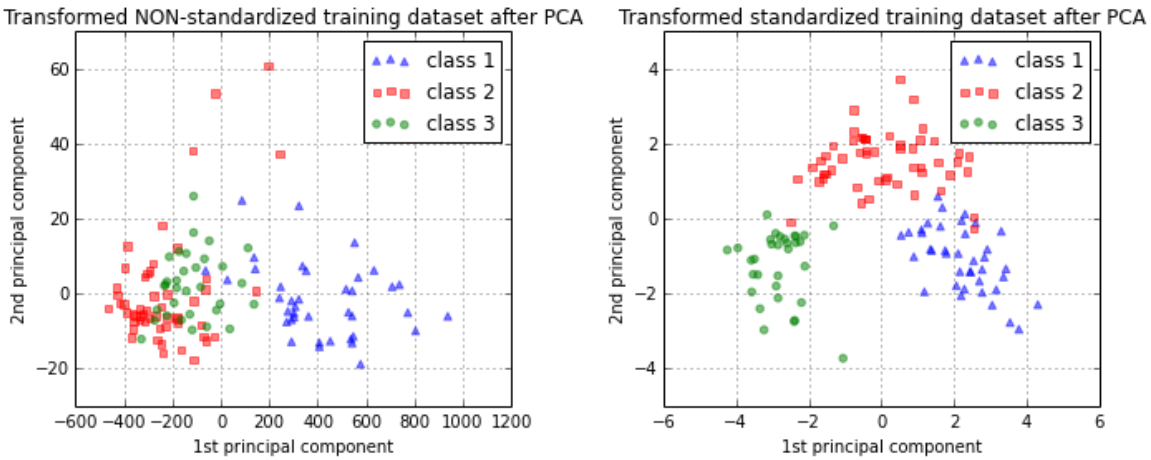

Ilustración 13: Ilustración del feature scaling o normalizado de valores [14]

En la imagen superior podemos apreciar cómo afecta el escalado de los atributos en un modelo de clasificación. La parte izquierda muestra como la primera componente (o atributo) oscila entre -600 y 1200, mientras que la segunda componente oscila entre 0 y 80, la cual cosa hace que la clasificación no se realice correctamente.

En la parte derecha se observa como las dos principales componentes están en la misma escala, de tal manera que se realiza la clasificación correctamente y las diferentes clases se pueden diferenciar.

Esto se puede realizar con la clase StandardScaler de SCI-KIT Learn.

Una vez escalados los datos procedemos a entrenar una SVM con varios kernels para evaluar su precisión.

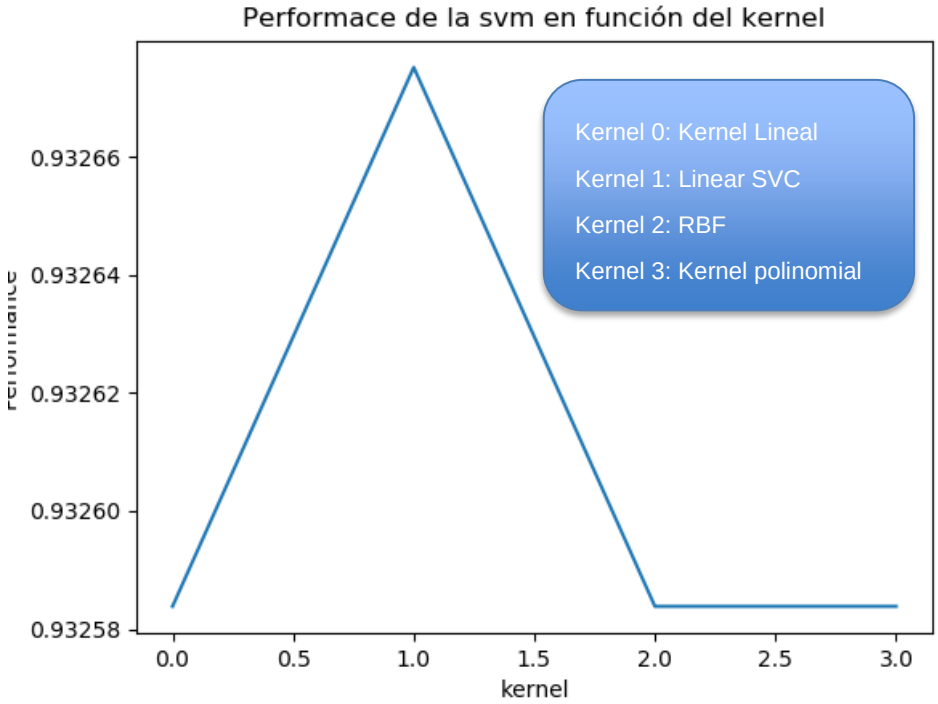

Ilustración 14: Precisión de los diferentes Kernel

Podemos observar como el resultado se sitúa de media por encima de un **93%**, parece razonable.

Observamos como el mejor kernel a nivel de precisión es el Linear SVC, el cual es exactamente igual que el lineal, pero está optimizado para grandes volúmenes de datos.

Por lo que se refiere al kernel polinomial se ha escogido de grado 3, y vistos los resultados un grado alto no resulta apropiado.

Finalmente, para el kernel RBF se han escogido unos parámetros estandarizados. Esto se debe a que, si los parámetros de partida no están determinados, la función que realiza el entreno tarda mucho a converger y, por tanto, a entrenar el modelo.

La variación en la precisión es casi inexistente, la cual cosa nos indica que este no es el modelo más apropiado de tal manera que no se ha indagado más en la elección de los hiperparametros.

Procedamos con un *árbol de decisión:*

En este caso a priori no se ha requerido ningún pre proceso no descrito anteriormente, de tal manera que se procede directamente a la evaluación de la precisión del modelo.

En primera instancia limitamos la profundidad del árbol de decisión. No hay una manera estandarizada de hacerlo ya que dependerá de la cantidad de atributos y de la relevancia de estos.

Con la profundidad totalmente des limitada, se obtiene una puntuación de **93,19%.**

Si limitamos la profundidad a 3, también obtenemos la misma puntuación. A priori con solo 3 niveles debería empeorar dada la poca diversificación del árbol.

Parece una puntuación más que decente, nos fijaremos en el vector de predicción y en el original para ganar más “insights”.

Se trata de un vector de una longitud 273.970, en el cual tenemos un 93% de 0’s y un 7% de 1’s.

Si observamos el vector de test tiene una longitud de 68.000 muestras y también tiene un 7 % de 1’s, lo cual indica que el “Split” entre los dos “datasets” es coherente.

Por lo que se refiere al vector de predicción tiene una longitud de 68.000 muestras y un 0,02% de 1’s. Esto nos indica que la estructura del árbol no es correcta, ya que un vector solo de 0’s tendría una precisión de un **93%** en nuestro modelo.

Investigando un poco, se observó en [8] como el comportamiento de los árboles de decisión con “one-hot encoding” no es el óptimo.

Este efecto sucede dada la manera del árbol de decisión de escoger los atributos y su importancia. Como se ha explicado en el capítulo anterior, el algoritmo va dividiendo el “dataset” y escogiendo los atributos más relevantes en estas divisiones.

Si se trata de una variable continua el algoritmo podrá escoger entre varios valores, y el árbol crecerá en ambas direcciones.

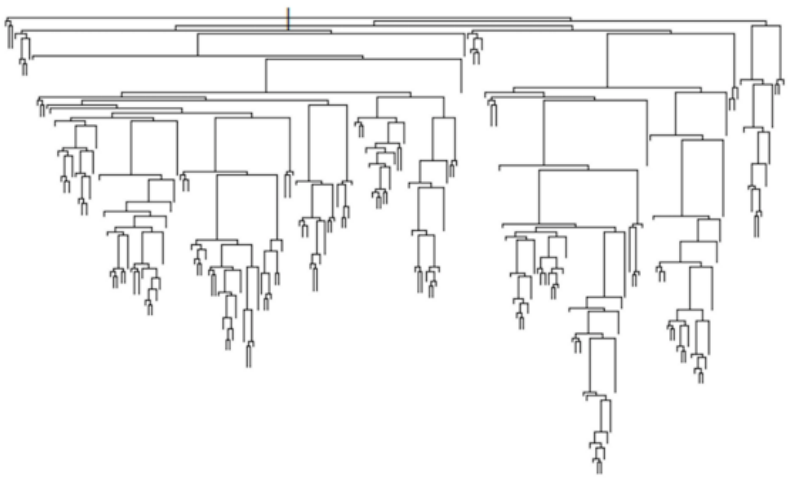

Ilustración 15: Estructura de un árbol de decisión correcto [8]

En caso de tener variables binarias resultantes del “encoding”, tendrán pocas opciones para dividirse, de tal manera que el árbol de decisión tiende a crecer en solo una dirección porque en cada división solo hay 1’s o 0’s. Un ejemplo a continuación:

Supongamos el siguiente “dataset”:

A	B	C	D
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Tabla 2: Estructura de un dataset no apto para realizar One-hot Encoding en árboles de decsión

El algoritmo a seguir para subdividir el “dataset” sería:

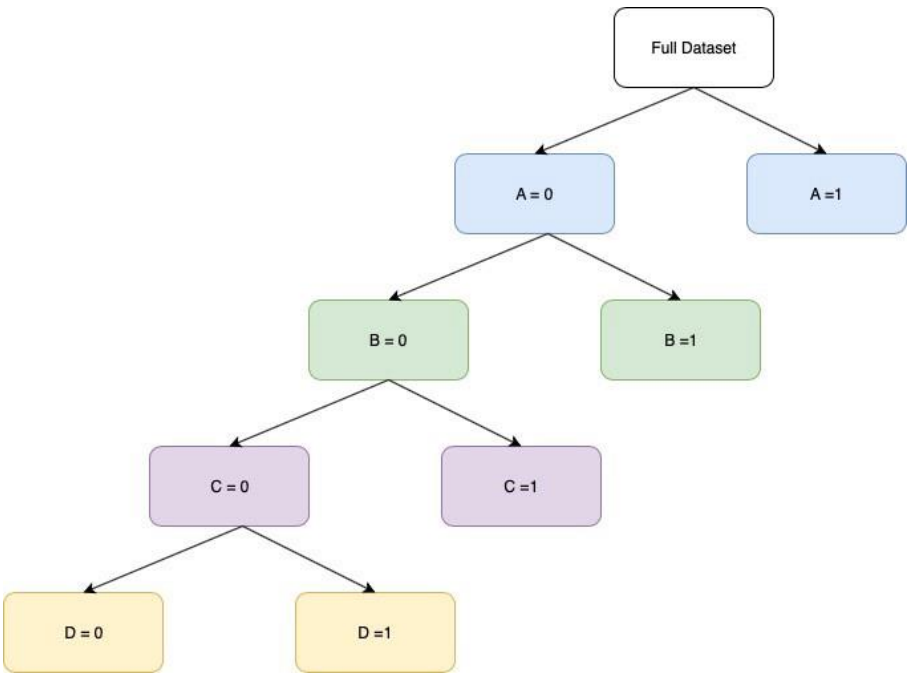

Ilustración 16: Algoritmo de subdivisión de un Dataset con One-hot encoding [8]

Para remediarlo hay que buscar una nueva manera de realizar el “encoding”, que sea de carácter numérico y que no confunda al algoritmo como podría suceder con una asignación “string” – numero.

Así pus se decide utilizar el target “encoding”. Supongamos que tenemos una variable no numérica x, la cual tiene asociado un resultado y. Para cada elemento de x se calcula la

media de los correspondientes valores de y. Posteriormente se reemplaza cada valor de x con su correspondiente media.

Con este tipo de "encoding" la precisión mejora ligeramente, se sitúa sobre un **94,5%.**

Pero, lo más importante es que la cantidad de 1's ha aumentado drásticamente y tiene una proporción muy parecida a la del training set, de tal manera que igual no se ha logrado un incremento drástico de puntuación, pero si a nivel de consistencia de datos.

Añadir que esta puntuación se ha obtenido sin limitar la profundidad del árbol, ya que en este caso nos interesa ser todo lo exactos que se pueda dada la poca cantidad de 1's en nuestro "dataset". Esta medida puede ser que introduzca cierto "overfitting", de tal manera que con el siguiente modelo se intentará reducirlo.

A continuación, se trabajará con un Random Forest.

Se entiende pues, que este modelo sigue la dinámica del árbol de decisión, de manera que para maximizar resultados se utilizará el "Target Encoder".

Se pueden ajustar multitud de parámetros tales como el número de árboles, el máximo número de atributos considerados para el algoritmo de entreno, la profundidad de cada carbol….

No limitaremos ni los atributos ni la profundidad de cada árbol dado que nos interesa la mayor precisión posible. Por lo que se refiere al número de árboles parece una buena idea según la teoría de "ensemble learning" establecer un numero considerablemente alto para tener un número mayor de "voters", reducir "overfitting" y aumentar precisión.

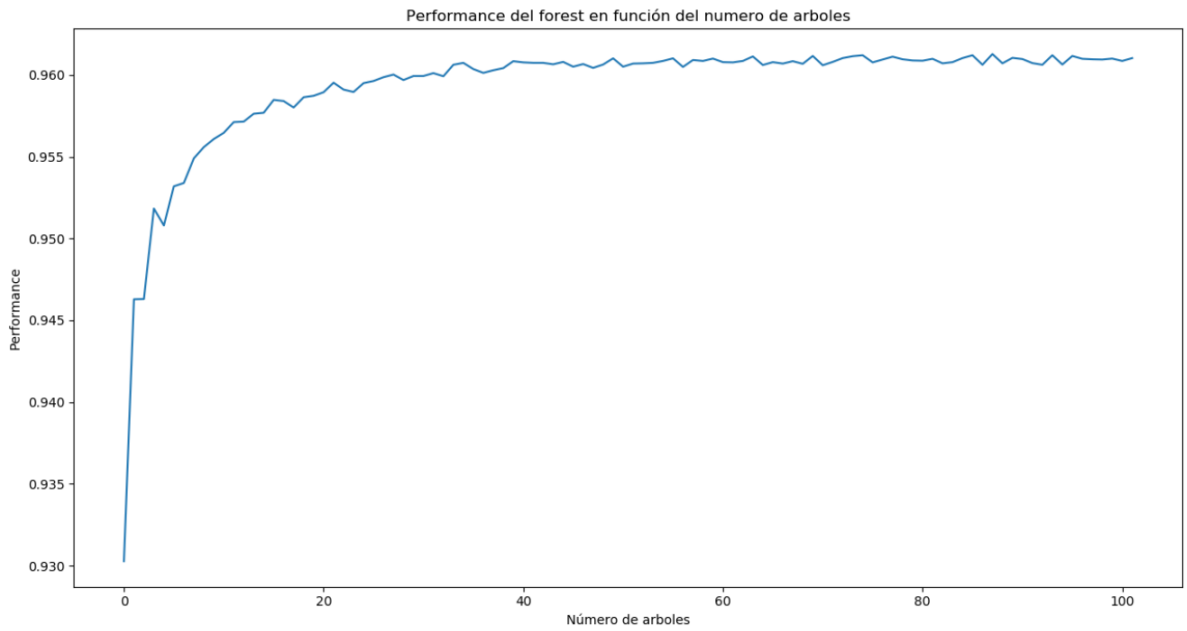

Ilustración 17: Random Forest según número de árboles aplicado al caso 1

Como vemos en la figura superior a partir de 40 árboles se llega a la puntuación máxima. También podemos observar como la precisión de este modelo es de un **96%,** *declarando este modelo como el mejor para este caso***.**

4.2. Vehículos de los clientes

Dentro de cartera de clientes de la marca, hay un sector de clientes los cuales llevan comprando vehículos durante muchos años.

El objetivo de este caso es poder hacer ofertas personalizadas a este segmento de clientes para que renueven su vehículo actual pasado un determinado tiempo. Este tiempo de media suele ser unos 10 años, de tal manera que se extraerá un "dataset" con estas personas.

En un contexto normal, ejecutaríamos una matriz de correlación y escogeríamos aquellos atributos más relevantes, pero en este caso y por imposición del fabricante se han escogido los siguientes atributos:

- **Relación con el vehículo:** Atributo no numérico que indica si el vehículo está a nombre del comprador o de su conductor habitual
- **Relación Activa:** Atributo que indica si el vehículo pertenece al cliente
- **Tipo de cliente:** Atributo que indica si el cliente es particular, empresa, autónomo..
- **Contactable:** Atributo que indica si el cliente es contactable
- **Financiación:** Atributo que indica si el vehículo está financiado
- **Tipo de persona:** Atributo no numérico que indica si se trata de una persona física o una empresa.
- **Sexo:** Atributo no numérico que indica el sexo del cliente

Como hemos visto en la descripción del caso, el número de atributos en este es muy superior a los otros dos planteados. Parece cuestionable la relevancia de algunos atributos en la predicción de manera que se introduce el concepto de reducción dimensional.

Se parte de un "dataset" con 411.000 registros, y una vez realizado el pre proceso tenemos un número final de 37.463 registros. Observamos cómo ha bajado drásticamente el número debido a la mala información de algunos atributos, cosa que podría influir en el rendimiento de nuestros modelos por "underfitting"

Si realizáramos el entrenamiento del modelo con todos los atributos estos podrían hacer difícil que el modelo encontrara patrones en nuestro "training set". Además, el proceso de entrenamiento sería mucho más lento.

Para realizar reducción dimensional en este caso se utilizará el método PCA o "principal component análisis" el cual es el método más utilizado. Entendemos pues como un "dataset" es una representación espacial de n dimensiones, tantas como atributos.

Este método identifica el hiperplano más cercano a los datos, de tal manera que proyecta los datos en éste.

Antes de proyectar estos datos, hay que escoger el hiperplano adecuado.

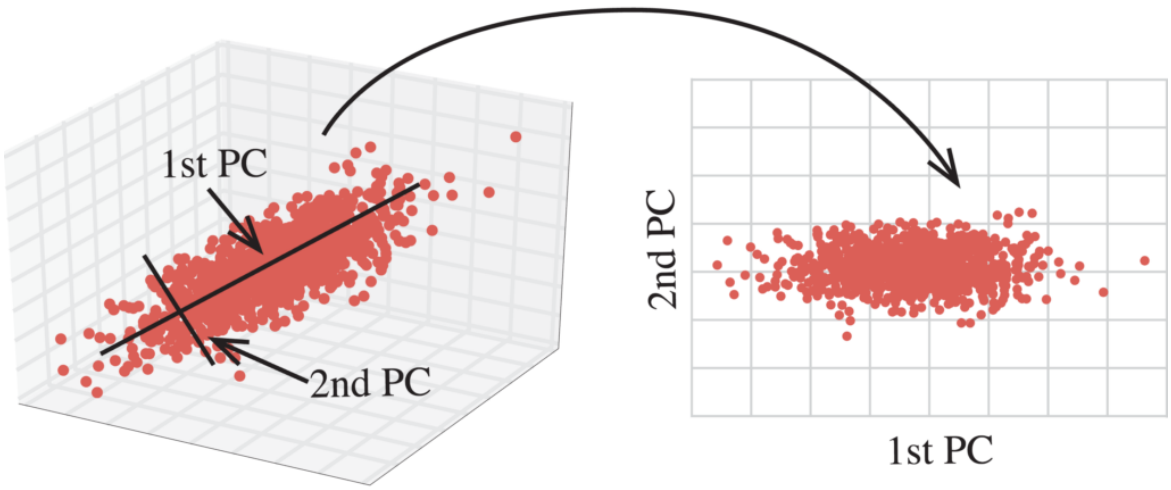

Ilustración 18: Ejemplo de reducción dimensional [6]

Como vemos en la imagen superior, pasamos de un espacio de 3 dimensiones a un espacio de 2 dimensiones. Se han escogido las dos proyecciones que tienen más varianza, de tal manera que se perderá menos información que con otras proyecciones. Otra manera de justificarlo sería que es el eje que minimiza el MSE de la distancia entre el "dataset" original y la proyección en dicho eje.

El "PCA" se puede implementar con la librería SCIKIT-Learn y su método PCA.

Antes de implementarlo, se puede obtener la varianza de cada componente,

```
>>> pca.explained_variance_ratio_
array([9.87736068e-01, 6.59679322e-03, 4.96250424e-03, 4.43930985e-04,
       2.18254350e-04, 1.28769604e-05, 8.56104946e-06, 6.70191117e-06,
       5.57409152e-06, 4.22351347e-06, 2.16516456e-06, 1.46165235e-06,
       7.55880618e-07, 1.09370883e-07, 1.83255927e-08, 1.47436221e-09,
       1.26145275e-48])
```

Ilustración 19: Varianza de cada componente de nuestro "Dataset"

En la imagen superior podemos observar la proporción de la varianza del "dataset" en cada eje de los componentes.

Podemos observar cómo el 98% de la varianza del "dataset" está en el primer eje, mientras que un 0,06% en el segundo. Es razonable imponer que los otros ejes no aportan información relevante y sólo se trabaje con el primero, de tal manera que el parámetro de nuestro PCA será de un 98% (para no incluir el resto de ejes).

Se plantea para empezar una *SVM:*

Se aplica escalado y codificación de tipo “target encoding”

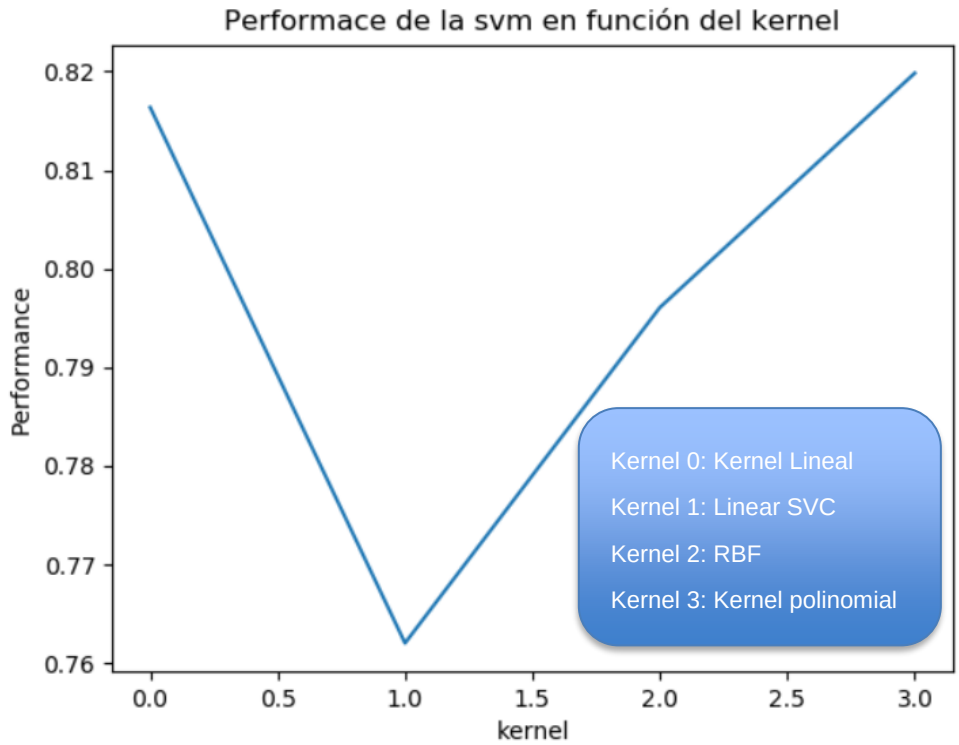

Ilustración 20: Precisión de la SVM en función del Kernel aplicado al caso 2

Podemos apreciar como en los resultados los “kernels” que funcionan mejor son el Lineal y el polinomial. Este último es de orden 3 y la similitud de precisión con el lineal sugiere que un orden mayor no sería necesario e introduciría “overfitting”.

La mejor precisión obtenida con una SVM es de un **82%**.

A continuación, se plantea un *Árbol de decisión*:

Una vez visto en el caso anterior que el “one-hot encoding” no funcionaba correctamente con este modelo, se procede a utilizar “target encoding”.

También trabajaremos sin reducción dimensional y con reducción dimensional, la cual prácticamente no afectará a la precisión de nuestro modelo y simplificará el entreno.

Para determinar la profundidad del árbol, nos ayudamos del siguiente gráfico donde aparecen los dos casos.

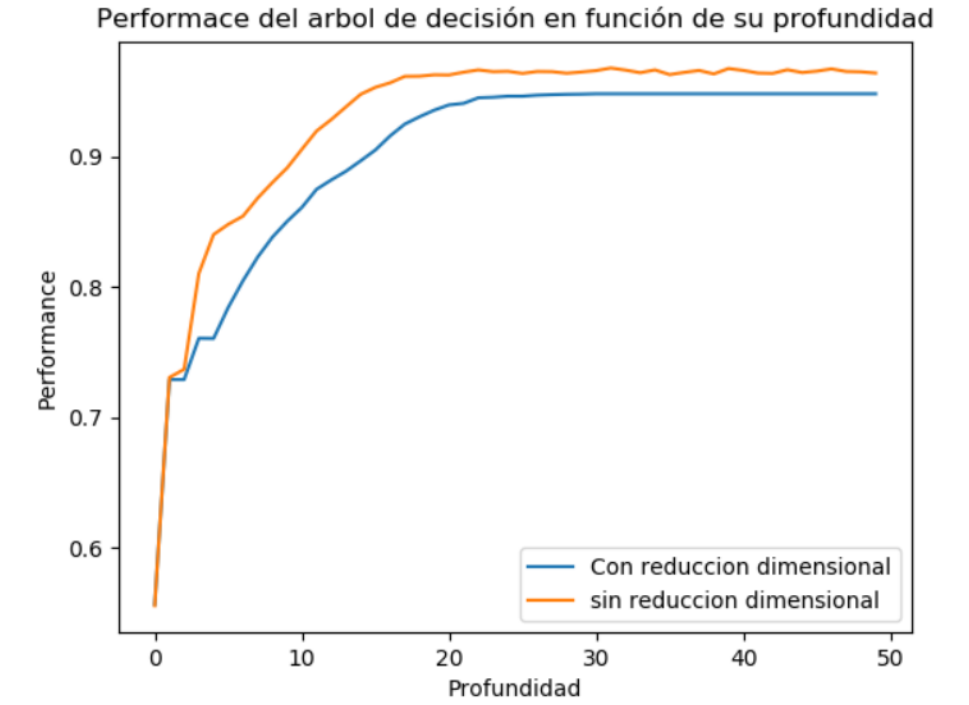

Ilustración 21: Precisión del árbol en función de su profundidad aplicado al caso 2

Consideramos pues, que una profundidad superior a 25 sería suficiente para obtener la máxima puntuación posible. Con reducción dimensional la máxima precisión es de un **94,5 %** mientras que sin, la puntuación máxima es de un **96,3%**.

Si no tenemos ninguna restricción a nivel hardware podemos afirmar que la mejor opción es sin reducción dimensional, ya que estamos prácticamente en **2%** por encima. En su defecto, se trabajaría con reducción ya que la puntuación es más que decente.

Finalmente, se trabajará con un *"Random Forest":*

Como en el árbol de decisión, se trabajará con un "target encoder". No se limitará ningún hiperparámetro excepto el número de árboles, de manera que el modelo será muy preciso sin demasiado "overfitting" gracias a que se realiza mediante "ensemble learning". A continuación, compararemos el número de árboles con y sin reducción dimensional.

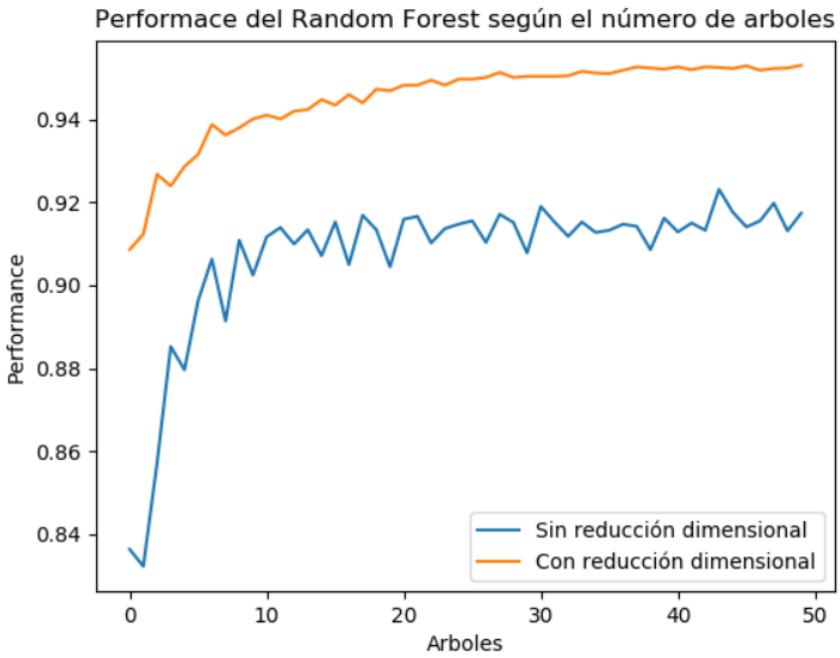

Ilustración 22: Precisión del "Random Forest" con y sin reducción dimensional aplicado al caso 2

En la ilustración superior queda demostrado como un gran número de dimensiones o atributos puede llegar a afectar el modelo. Limitar los atributos en un solo árbol empeora ligeramente su precisión, mientras que en un modelo que utiliza "ensemble learning" resulta beneficioso. Así pues, dada una gran cantidad de atributos un modelo basado en "ensemble learning" no funciona correctamente.

Como conclusión del caso, el modelo a utilizar sería un *"Random Forest" con reducción dimensional y limitado a 30 árboles* obteniendo una precisión de un **96%**.

Esta elección puede parecer errónea dado que el árbol de decisión sin reducción dimensional tiene una mayor puntuación, pero posiblemente se ajuste demasiado al "training set", es decir, tenga demasiado "overfitting". Utilizando el "Random Forest" tendremos una puntuación ligeramente inferior, pero nos aseguraremos de ser más polivalentes.

4.3. Visitas por taller de los clientes

Desde hace unos años las visitas por talleres oficiales de la marca se están reduciendo de manera considerable. Con este caso se pretende identificar patrones en las características de los clientes y así poder impactar con promociones a aquellos que tienen altas posibilidades de no pasar nunca por taller.

Según criterios establecidos un cliente resulta fiel si cada año pasa almenos una vez por taller, de tal manera que todos aquellos clientes que no cumplan esta condición serán considerados infieles y se les impactará con promociones varias.

En un contexto normal, ejecutaríamos una matriz de correlación y escogeríamos aquellos atributos más relevantes, pero en este caso y por imposición del fabricante se han escogido los siguientes atributos:

- **Provincia:** Atributo no numérico que indica la provincia dónde reside el cliente
- **Sexo:** Atributo no numérico que indica el sexo del cliente
- **Tipo de Persona:** Atributo no numérico que indica si se trata de una persona física o una empresa.
- **Relación con el vehículo:** Atributo no numérico que indica si el vehículo está a nombre del comprador o de su conductor habitual
- **Modelo del vehículo:** Atributo no numérico que indica el modelo del vehículo
- **Flag Orden de reparación:** Atributo numérico que determina si se ha realizado una orden de reparación en este último año.

Este caso parte con el “dataset” más pequeño de todos, con 17.836 registros iniciales.

Una vez hecho el pre proceso, los registros disminuyen a 16.865. Es de lejos el “dataset” con los atributos mejor informados de los 3 casos, pero su cantidad de registros es realmente mínima dados sus 6 atributos. Esto como se ha comentado en el caso anterior podría afectar a la precisión de los modelos.

Se empieza por una *SVM:*

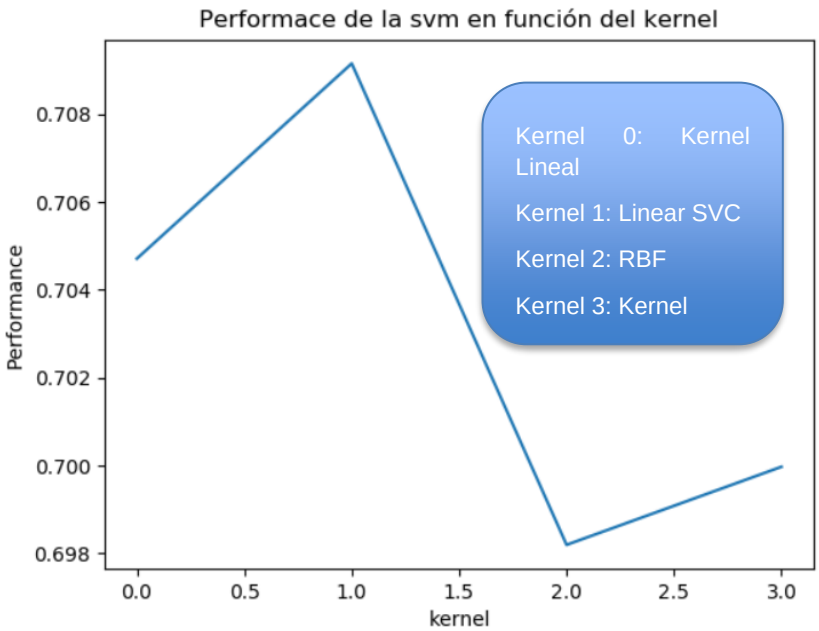

Ilustración 23: Precisión de una SVM con varios kernels aplicada al caso 3

Como podemos observar en la figura superior la mayor precisión del kernel que se ajusta más (Linear_SVC) es de un **71%.** Las otros kernels le siguen muy de cerca, de tal manera que vista la baja puntuación podemos extraer dos conclusiones:

- Una SVM no es un modelo adecuado para este caso dada la baja puntuación y la poca diferencia entre kernels
- Estamos en una situación de “underfitting” por las mismas razones que en la anterior conclusión

El siguiente paso es aplicar un *Árbol de decisión*:

Pasamos a utilizar directamente "target encoding" de tal manera que quedará determinar la profundidad de este para obtener la mayor precisión:

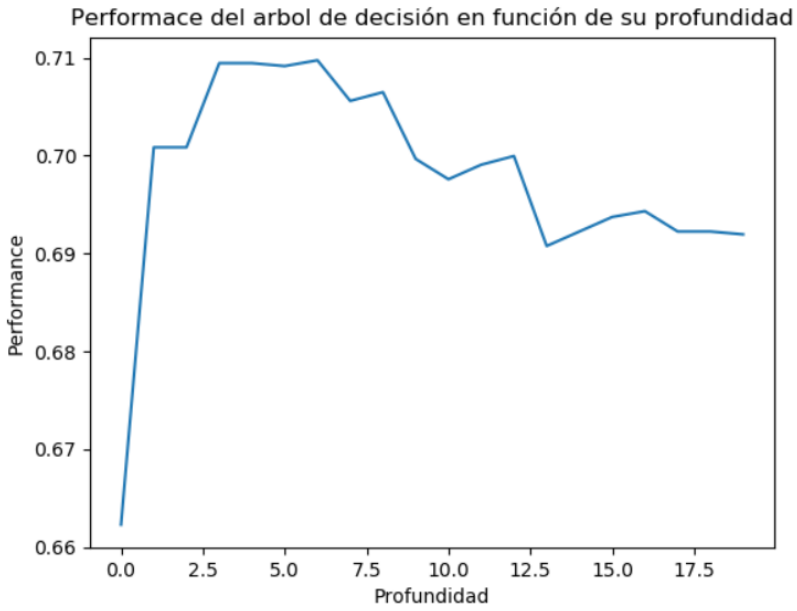

Ilustración 24: Relación Profundidad- Precisión de un árbol aplicado al caso 3

En este caso podemos observar como una mayor profundidad no se obtiene un mejor resultado. También, la puntuación más alta, que se consigue una profundidad de entre 3 y 5, es muy parecida a la de la SVM. Esto podría ser un claro indicativo de "underfitting" ya que dos modelos diferentes, tienen una precisión muy parecida.

Finalmente, se plantea un *"Random Forest":*

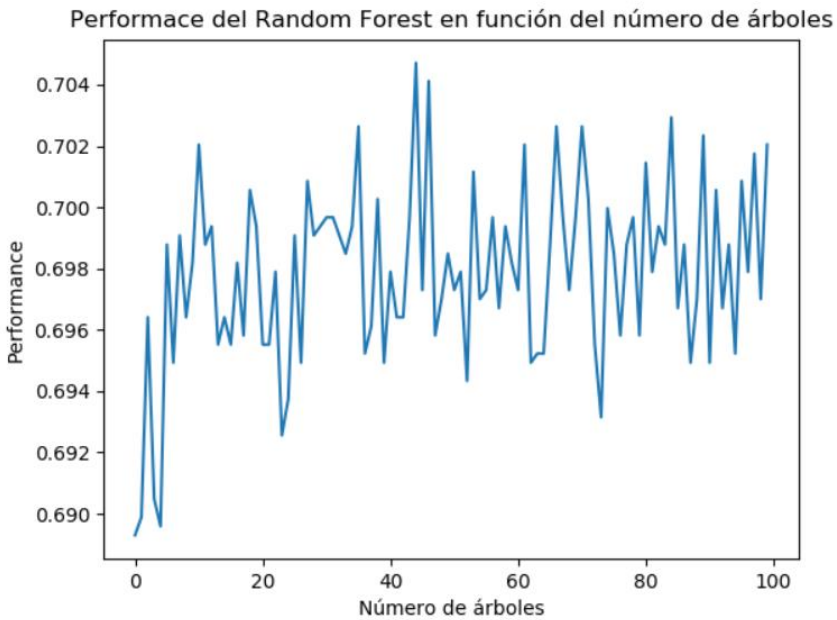

Ilustración 25: Precisión del Random Forest aplicado al caso 3

En la figura superior se puede apreciar como la precisión es aleatoria a partir de unos 10 árboles oscilando entre **69,4%** y **70,4%**. Además, y comparando con los anteriores dos modelos la precisión resulta ser muy parecida. Llegados a este punto podemos afirmar:

- Que con los 3 modelos estudiados se obtenga una puntuación tan similar nos indica que hay déficit de muestras o bien, falta de correlación entre atributos
- Estamos totalmente limitados por el número de registros, o siendo más generales, por el "dataset" del que partimos de manera que ningún modelo superaría las precisiones ya adquiridas
- La única manera de obtener una mayor precisión sería introducir datos de años anteriores o bien, determinar nuevos atributos los cuales introdujeran más relevancia al "dataset"

Tabla resumen de los resultados obtenidos:

Caso	SVM	Árbol de decisión	Random Forest
1	93,2%	94,5%	96%
2	82%	96,3%	96%
3	71%	71%	70,4%

Tabla 3: Resumen de las precisiones de cada modelo según el caso

*Se han considerado los mejores casos para cada modelo

5. Presupuesto

Este proyecto es totalmente de software, de manera que los costes se limitan al sueldo de los ingenieros y al precio del software utilizado durante todo el desarrollo.

Definiremos dos integrantes para el proyecto, Ingeniero junior (Autor) e Ingeniero senior (supervisor). Por lo que se refiere a la dedicación, el ingeniero junior ha dedicado 20 horas semanales, mientras que el ingeniero senior ha dedicado 2 horas por semana.

El único software de pago que se ha utilizado es el IDE de Python, PyCharm, el cual se paga en forma de suscripción mensual. La librería SCIKIT-Learn es totalmente gratuita.

La duración del proyecto ha sido de 4 meses, de tal manera que la distribución de los gastos queda:

Perfil	Cantidad	Precio/Hora	Horas/Mes	Coste
Ingeniero Junior	1	10€	80	3200€
Ingeniero Senior	1	25€	8	800€

Tabla 4: Coste a nivel de personas

Producto	Cantidad	Precio/Mes	Número de meses	Coste
PyCharm IDE	2	19,90€	4	79,6€

Tabla 5: Coste a nivel de Software

Coste total del proyecto: 4079,6 €

6. Conclusiones y futuro desarrollo:

El objetivo de esta tesis era obtener unas predicciones los suficientemente precisas para poder impactar a diversos grupos de personas según el interés del fabricante.

En los dos primeros casos, se puede afirmar que el objetivo se ha logrado, con precisiones por encima del 95% en ambos casos y a priori, limitando el “overfitting”.

Estos impactos se realizarían mediante e-mail marketing y se podrían lanzar posteriormente a las campañas publicitarias, para tener una mayor repercusión.

Por lo que se refiere a aquellas personas que estén en el 5 % de error y si muestran enfado por haber sido impactadas, se realizaría una lista de personas descontentas, la cual una vez llegada a un número considerable de personas se podría realizar un modelo predictivo para no tenerlas en cuenta desde un buen principio.

Por lo que se refiere el caso 3 la precisión es muy baja y sería imposible realizar impactos con garantías de que la persona impactada cumple las condiciones deseadas.

Para mejorar la precisión de este último caso se deberían coger datos mucho anteriores, arriesgándonos a perder relevancia sobre estos por ser las condiciones tan cambiantes año tras año.

Otra manera sería mejorar la base de datos, añadiendo datos de mayor relevancia y orientar el contenido de la base de datos a realizar predicciones.

Para finalizar y como futuro desarrollo, se podrían buscar modelos más complejos y utilizar técnicas de “boosting” para los arboles de decisión y los “Random forest”. Por lo que se refiere a la SVM se podrían buscar kernels que se adaptaran mejor a nuestros datos, incluso crear nuestro propio kernel, aunque haya sido el modelo que “peor” haya funcionado en todos los casos.

Además, se podrían implantar redes neuronales para mejorar las predicciones.

Si se siguiera con la misma base de datos esto sería irrealizable en los casos 2 y 3 en los cuales escasean los datos, ya que en estos sistemas se puede afirmar que para prácticamente todos los casos cuantas más muestras, mejor precisión.

Bibliografía:

[1] Michela Banko, Eric Brill “Scaling to very very large corpora for natural Language Disambiguation”. Microsoft [Online] Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/acl2001.pdf [Accesed: 12 April 2019]

[2] Dietterich, Thomas. (1995). Overfitting and Undercomputing in Machine Learning. ACM Comput. Surv. 27. 326-327. 10.1145/212094.212114.

[3] David Wolpert, William Macready “No free Lunch Theorems for optimization” NASA [Online] Available: https://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf [Accesed: 20 April 2019]

[4] Alexandru Niculescu-Mizill, Rich Caruana “Predicting Good Probabilities with Supervised Learning” [Online] Available: https://www.cs.cornell.edu/~alexn/papers/calibration.icml05.crc.rev3.pdf [Accesed: 20 May 2019]

[5] Andreas Muller, Sarah Guido. *Introduction to machine learning: A guide for data scientists*, 1st ed. Sebastopol, USA: O’Reilly Media, 2017

[6] Aurélien Géron. Hands on Machine Learning with Scikit Learn and TensorFlow, 1st ed. Sebastopol, USA: O’Reilly Media, 2017

[7] Jake VanderPlas. *Python Data Science Handbook, 1st ed.* Sebastopol, USA: O’Reilly Media, 2017

[8] Rakesh Ravi “One-Hot Encoding is making your Tree-Based Ensembles worse, here’s why” [Online] Available:https://towardsdatascience.com/one-hot-encoding-is-making-your-tree-based-ensembles-worse-heres-why-d64b282b5769 [Accessed: 30 may 2019]

[9] https://jingwen-z.github.io/training-set-and-test-set/

[10] https://originsymwisedownload.symantec.com/resources/webguides/contentanalysis/22/Content/Topics/Concepts/aml_process.htm

[11] https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76)

[12] https://medium.com/@athif.shaffy/one-hot-encoding-of-text-b69124bef0a7

[13] https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwiGq_LQvd_iAhVF5uAKHRwICJwQjRx6BAgBEAU&url=https%3A%2F%2Ftowardsdatascience.com%2Fsupport-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47&psig=AOvVaw2fc_T74UmjqXq8DVxD4NvD&ust=1560275571985124

[14] https://sebastianraschka.com/Articles/2014_about_feature_scaling.html

Apéndice:

A continuación, se mostrará el código utilizado para cada caso:

Caso 1

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd
import csv
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
import numpy as np
from sklearn.model_selection import ShuffleSplit
import matplotlib.pyplot as plt

#------Arbol de decisión con OneHot Encoding
data = pd.read_csv('leads_pedidos_conmodelos.csv', encoding="latin1",
error_bad_lines=False, quoting=csv.QUOTE_NONE, sep=';',
low_memory=False)
corr=data.corr()
data=data.dropna()
data_dummies=pd.get_dummies(data)
features= data_dummies.ix[:,
'MODELO_ARTEON':'DESC_TOKEN_VW_YMW_INT_EVENTS']
X=features
y=data_dummies['FLAG_COMPRADO'].values
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)
score= []
for i in range(20):
    tree_clf=DecisionTreeClassifier(max_depth=i+1)
    tree_clf.fit(X_train, y_train)
    prediccion=tree_clf.predict(X_test)
    score.insert(i, accuracy_score(y_test, prediccion))

plt.plot(score)
plt.title('Performace del arbol de decisión en función de su
profundidad')
plt.ylabel('Performance')
plt.xlabel('Profundidad')

tree_clf = DecisionTreeClassifier()
tree_clf.fit(X_train, y_train)
prediccion = tree_clf.predict(X_test)
score.insert(i, accuracy_score(y_test, prediccion))

export_graphviz(tree_clf, out_file=("test1.dot"), feature_names=
X.columns, rounded=True, filled=True)
prediccion=tree_clf.predict(X_test)
print("la precison del modelo es de :", accuracy_score(y_test,
prediccion))

#Svm (Aprovechamos el split del dataset y el enoding anterior)
from sklearn import svm
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X,y)
X_c=scaler.fit_transform(X,y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)
C = 1
Punt=[]
i=1
models = (svm.SVC(kernel='linear', C=C),
          svm.LinearSVC(C=C, max_iter=10000),
          svm.SVC(kernel='rbf', gamma=0.7, C=C),
          svm.SVC(kernel='poly', degree=3, gamma='auto', C=C))
for clf in models:
    clf.fit(X_train, y_train)
    prediccion=clf.predict(X_test)
    print("la precison del modelo es:", accuracy_score(y_test,
prediccion))
    Punt.insert(i, accuracy_score(y_test, prediccion))
    i+=1

plt.plot(Punt)
plt.title('Performace de la svm en función del kernel')
plt.ylabel('Performance')
plt.xlabel('kernel')

##-----Arbol de decisión con target encoding

data = pd.read_csv('leads_pedidos_conmodelos.csv', encoding="latin1",
error_bad_lines=False, quoting=csv.QUOTE_NONE, sep=';',
low_memory=False)
import category_encoders as ce
encoder=ce.TargetEncoder()
X=data.drop('FLAG_COMPRADO', axis=1)
y=data.FLAG_COMPRADO
encoder.fit(X,y)
X=encoder.transform(X,y)
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)
score= []
for i in range(20):
    tree_clf=DecisionTreeClassifier(max_depth=i+1)
    tree_clf.fit(X_train, y_train)
    prediccion=tree_clf.predict(X_test)
    score.insert(i, accuracy_score(y_test, prediccion))

plt.plot(score)
plt.title('Performace del arbol de decisión en función de su
profundidad')
plt.ylabel('Performance')
plt.xlabel('Profundidad')

export_graphviz(tree_clf, out_file=("test1.dot"), feature_names=
X.columns, rounded=True, filled=True)
prediccion=tree_clf.predict(X_test)
print("la precison del modelo es de :", accuracy_score(y_test,
prediccion))

#----Random forest (Aprovechamos el split del dataset y el enoding
anterior)

from sklearn.ensemble import RandomForestClassifier
score= []
for i in range(500):
```

```
    rnd_clf= RandomForestClassifier(n_estimators=i+1)
    rnd_clf.fit(X_train, y_train)
    prediccion=rnd_clf.predict(X_test)
    print("la precison del modelo bosque es de  :",
accuracy_score(y_test, prediccion))
    score.insert(i, accuracy_score(y_test, prediccion))
plt.plot(score)
plt.title('Performance del forest en función del numero de arboles')
plt.ylabel('Performance')
plt.xlabel('Número de arboles')
```

Caso 2

```
import pandas as pd
import csv
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
from sklearn.model_selection import cross_val_score
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.tree import export_graphviz
from copy import copy, deepcopy

data = pd.read_csv('TFGdatasetpetit.csv', encoding="latin1",
error_bad_lines=False, quoting=csv.QUOTE_NONE, sep=';')
data=data.drop_duplicates(subset='COD_VEHICULO_CLIMAPRO', keep=False)
data=data.dropna()
#amb encoder tipus target
import category_encoders as ce
encoder=ce.TargetEncoder()
aux=copy(data.EDAD)
data=data.drop('EDAD', axis=1)
features=data.ix[:,(data.columns)]
X=features.values
y=aux
X=encoder.fit_transform(X,y)
modelos=deepcopy(X[15])
aux=deepcopy(y)
X=X.drop(X.columns[15], axis=1)
modelos=modelos.to_numpy()
aux=aux.to_numpy()
X.insert(loc=21, column=22, value=aux, allow_duplicates=True)
y=modelos
pca=PCA(n_components=0.95)

#SVM con reduccion diemensional
from sklearn.svm import LinearSVC
from sklearn import svm
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_C=scaler.fit_transform(X)
X_R=pca.fit_transform(X_C)
X_train, X_test, y_train, y_test = train_test_split(X_R, y,
train_size=0.8)
C=1
pca.fit(X)
```

```
Punt=[]
i=1
models = (svm.SVC(kernel='linear', C=C),
          svm.LinearSVC(C=C, max_iter=10000),
          svm.SVC(kernel='rbf', gamma=0.7, C=C),
          svm.SVC(kernel='poly', degree=3, gamma='auto', C=C))
for clf in models:
    clf.fit(X_train, y_train)
    prediccion=clf.predict(X_test)
    print("la precison del modelo es:", accuracy_score(y_test,
prediccion))
    Punt.insert(i, accuracy_score(y_test, prediccion))
    i+=1

plt.plot(Punt)
plt.title('Performace de la svm en función del kernel')
plt.ylabel('Performance')
plt.xlabel('kernel')
#Arbol de decision con reducción dimensional
X_R=pca.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_R, y,
train_size=0.8)
score=[]
for i in range(50):
    tree_clf=DecisionTreeClassifier(max_depth=i+1)
    tree_clf.fit(X_train, y_train)
    #export_graphviz(tree_clf, out_file=("tfgdatasetpetit.dot"),
feature_names= features.columns, rounded=True, filled=True)
    prediccion=tree_clf.predict(X_test)
    print("la precison del modelo es de  :", accuracy_score(y_test,
prediccion))
    score.insert(i,accuracy_score(y_test, prediccion))

    plt.plot(score, label="Con reduccion dimensional")
    plt.title('Performace del arbol de decisión en función de su
profundidad')
    plt.ylabel('Performance')
    plt.xlabel('Profundidad')

#decision tree sin reducción dimensional
p=[]
X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.8)
for i in range(50):
    tree_clf=DecisionTreeClassifier(max_depth=i+1)
    tree_clf.fit(X_train, y_train)
    #export_graphviz(tree_clf, out_file=("tfgdatasetpetit.dot"),
feature_names= features.columns, rounded=True, filled=True)
    prediccion=tree_clf.predict(X_test)
    print("la precison del modelo clasificacion es de  :",
accuracy_score(y_test, prediccion))
    p.insert(i, accuracy_score(y_test, prediccion))

    plt.plot(p, label="sin reduccion dimensional")
    plt.title('Performace del arbol de decisión en función de su
profundidad')
    plt.ylabel('Performance')
    plt.xlabel('Profundidad')

#Random forest
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.8)
prueba= []
from sklearn.ensemble import RandomForestClassifier
for i in range(50):
    rnd_clf= RandomForestClassifier(n_estimators=i+1)
    rnd_clf.fit(X_train, y_train)
    prediccion=rnd_clf.predict(X_test)
    prueba.insert(i, accuracy_score(y_test, prediccion))
    print(i)

plt.plot(prueba, label="Sin reducción dimensional")
plt.title('Performace del Random Forest según el número de arboles')
plt.ylabel('Performance')
plt.xlabel('Arboles')

#random forest con reducción dimensional
X_R=pca.fit_transform(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_R, y,
train_size=0.8)
prueba2=[]
for i in range(50):
    rnd_clf= RandomForestClassifier(n_estimators=i+1)
    rnd_clf.fit(X_train, y_train)
    prediccion=rnd_clf.predict(X_test)
    prueba2.insert(i, accuracy_score(y_test, prediccion))
    print(i)

plt.plot(prueba2, label="Con reducción dimensional")
plt.legend
```

Caso 3

```
import pandas as pd
import csv
import category_encoders as ce
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
from sklearn.model_selection import cross_val_score
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.tree import export_graphviz

data = pd.read_csv('ORTFG190520.csv', encoding="latin1",
error_bad_lines=False, quoting=csv.QUOTE_NONE, sep=';')
data=data.dropna()
encoder=ce.TargetEncoder()
y=data['TIENE_OR']
X=data.drop('TIENE_OR', axis=1)
encoder.fit(X,y)
```

```
X=encoder.transform(X,y)

#---------------Arbol de decisión
X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.8, test_size=0.2)
score= []
for i in range(20):
    tree_clf=DecisionTreeClassifier(max_depth=i+1)
    tree_clf.fit(X_train, y_train)
    prediccion=tree_clf.predict(X_test)
    score.insert(i, accuracy_score(y_test, prediccion))

plt.plot(score)
plt.title('Performace del arbol de decisión en función de su
profundidad')
plt.ylabel('Performance')
plt.xlabel('Profundidad')



#-----------------------Random forest
from sklearn.ensemble import RandomForestClassifier
sc=[]
for i in range(100):
    rnd_clf= RandomForestClassifier(n_estimators=i+1)
    rnd_clf.fit(X_train, y_train)
    prediccion=rnd_clf.predict(X_test)
    sc.insert(i, accuracy_score(y_test, prediccion))
    print(i)
plt.plot(sc)
plt.title('Performace del Random Forest en función del número de
árboles')
plt.ylabel('Performance')
plt.xlabel('Número de árboles')

#---------------SVM con reduccion dimensional

from sklearn import svm
from sklearn.preprocessing import StandardScaler
pca=PCA(n_components=0.95)
scaler=StandardScaler()
X_C=scaler.fit_transform(X)
X_R=pca.fit_transform(X_C)
X_train, X_test, y_train, y_test = train_test_split(X_R, y,
train_size=0.8)
print(pca.explained_variance_ratio_)
C = 1
Punt=[]
i=1
models = (svm.SVC(kernel='linear', C=C),
          svm.LinearSVC(C=C, max_iter=10000),
          svm.SVC(kernel='rbf', gamma=0.7, C=C),
          svm.SVC(kernel='poly', degree=3, gamma='auto', C=C))
for clf in models:
    clf.fit(X_train, y_train)
    prediccion=clf.predict(X_test)
    print("la precison del modelo es:", accuracy_score(y_test,
prediccion))
    Punt.insert(i, accuracy_score(y_test, prediccion))
    i+=1
```

```
plt.plot(Punt)
plt.title('Performace de la svm en función del kernel')
plt.ylabel('Performance')
plt.xlabel('kernel')
```